Izrada i programiranje edukativnog SCARA robota

Domjanović, Marko

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Marko Domjanović

Zagreb, 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentori:

Doc. dr. sc. Marko Švaco, mag. ing. mech.

Student:

Marko Domjanović

Zagreb, 2023.

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

Zahvaljujem se doc. dr. sc. Marku Švaci i mag.ing. Branimiru Ćaranu na dostupnosti, pristupačnosti, savjetima i pomoći prilikom izrade ovog završnog rada.

Zahvaljujem se svojoj obitelji, prijateljima i djevojci na podršci i strpljenju tijekom studiranja.

Marko Domjanović

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

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 – 04 / 23 – 6 / 1**	
Ur.broj: **15 - 1703 - 23 -**	

ZAVRŠNI ZADATAK

Student: **Marko Domjanović** JMBAG: **0035223136**

Naslov rada na hrvatskom jeziku: **Izrada i programiranje edukativnog SCARA robota**

Naslov rada na engleskom jeziku: **Designing and programming of an educational SCARA robot**

Opis zadatka:

SCARA roboti su vrsta industrijskih robota koji se često koriste u procesima rukovanja dijelovima i montaži dijelova manjih dimenzija i mase. SCARA roboti najčešće imaju četiri stupnja slobode gibanja, tri rotacije i jednu translaciju. Njihova primjena je česta u industrijama gdje je zahtijevana visoka brzinu i točnost ponavljanja. Osim u industriji, SCARA roboti prikladni su i u edukativnim svrhama zbog svoje jednostavnije mehaničke izvedbe od artikuliranih robota te matematički jednostavnijeg izvoda kinematike koja je nužna za osnovno upravljanje pozicijom vrha alata robota.

U sklopu ovog završnog rada potrebno je:

- Napraviti pregled i usporedbu postojećih edukativnih SCARA robota.
- Oblikovati edukativnog SCARA robota, za kojeg će se nestandardne mehaničke komponente moći izraditi koristeći dostupne 3D printere u Regionalnom centru izvrsnosti za robotske tehnologije (CRTA).
- Odabrati preostale standardne mehaničke komponente robota.
- Odabrati aktuatorski, senzorski i upravljački sustav za edukativnog SCARA robota.
- Sastaviti i funkcionalno ispitati rad svih sustava edukativnog SCARA robota.
- Primjenom softverskog ROS paketa MoveIt omogućiti pozicioniranje vrha alata edukativnog SCARA robota.
- Prema mogućnostima napraviti primjenu slijeđenja zadane putanje.

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

Zadatak zadan:
30. 11. 2022.

Datum predaje rada:
1. rok: 20. 2. 2023.
2. rok (izvanredni): 10. 7. 2023.
3. rok: 18. 9. 2023.

Predviđeni datumi obrane:
1. rok: 27. 2. – 3. 3. 2023.
2. rok (izvanredni): 14. 7. 2023.
3. rok: 25. 9. – 29. 9. 2023.

Zadatak zadao:
Doc. dr. sc. Marko Švaco

Predsjednik Povjerenstva:
Prof. dr. sc. Branko Bauer

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS OZNAKA

Oznaka	Jedinica	Opis
a_i	mm	Duljina segmenta
$\mathbf{a}$	-	Vektor djelovanja
$^{\mathbf{m}}\mathbf{A}_{\mathbf{n}}$	-	Matrica homogenih transformacija
d_i	mm	Udaljenost zglobova
$\mathbf{n}$	-	Vektor normale
$\mathbf{o}$	-	Vektor orijentacije
$\mathbf{p}$	-	Vektor položaja
p_x, p_y, p_z	-	Komponente vektora položaja
q_i	rad	Zakret i-te upravljane komponente
$\mathbf{q}$	-	Vektor unutarnjih koordinata
$\mathbf{r}$	-	Vektor vanjskih koordinata
ϑ	rad	Kut skretanja

SAŽETAK

U ovom se radu integrira edukativni SCARA robot s robotskim operativnim sustavom ROS. Cilj rada je oblikovati i sastaviti edukativni SCARA robot te uz ROS-ove integrirane softverske biblioteke i alate omogućiti upravljanje i pozicioniranje vrha alata robota. Komunikacijom između ROS-a i Arduina te ROS-ovog paketa MoveIt pomiču se zglobovi robota zadajući željene koordinate vrha alata unutar simulacijskog okruženja RViz.

Ključne riječi: ROS, SCARA, MoveIt, Python, RViz, Arduino

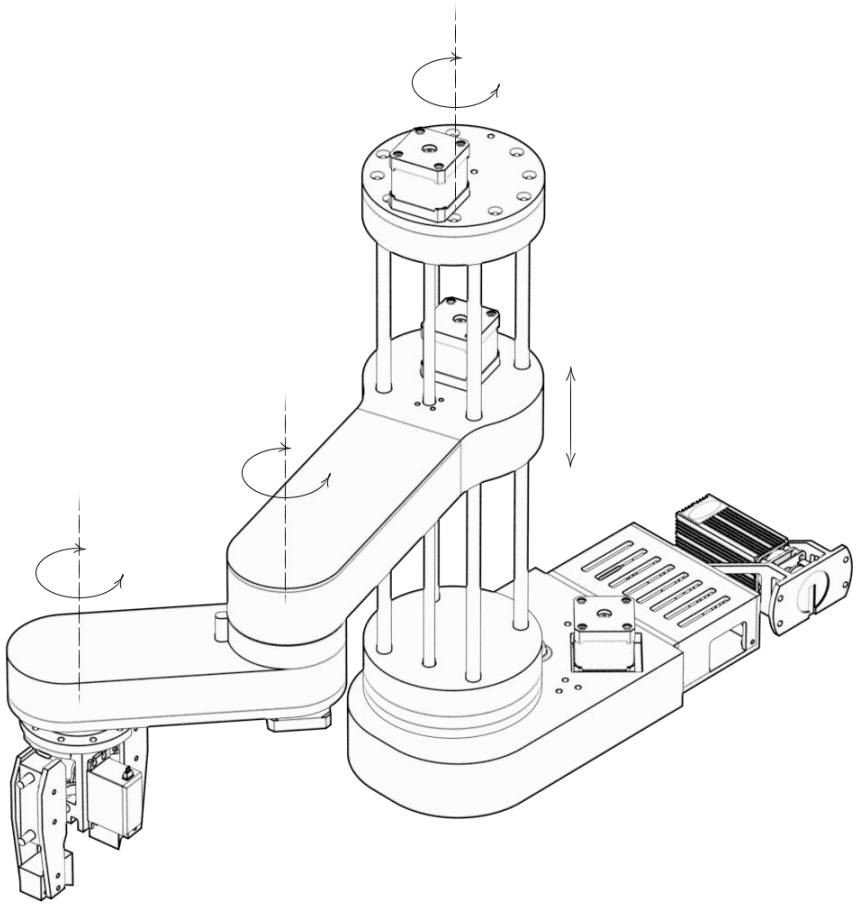
SUMMARY

In this work, an educational SCARA robot is integrated with the Robot Operating System (ROS). The goal of the work is to design and assemble an educational SCARA robot and, using ROS's integrated software libraries and tools, enable control and positioning of the robot's tool tip. Through communication between ROS and Arduino, as well as the ROS package MoveIt, the robot's joints are moved by specifying desired coordinates of the tool tip within the simulation environment RViz.

Key words: ROS, SCARA, MoveIt, Python, RViz, Arduino

1. UVOD

SCARA roboti su vrsta robotskih manipulatora s četiri stupnja slobode gibanja. Imitiraju ljudsku ruku tako da omogućuju rotaciju u svim zglobovima te okomitu translaciju. Za razliku od ljudske ruke, svi zglobovi SCARA robota su ograničeni na rotaciju oko vertikalne osi, odnosno nemaju mogućnost rotacije zglobova u svim smjerovima. Upravo ta jednostavnost izvedbe čini SCARA robote efikasnima u industrijskim postrojenjima radi brzine i točnosti ponavljanja koju pružaju.



Slika 1.1. Stupnjevi slobode SCARA robota

Glavna svrha SCARA robota je povećanje produktivnosti te se mogu koristiti u različitim operacijama, od „pick and place“, njihovog najčešćeg zadatka, sve do testiranja, sklapanja, pakiranja te inspekcije.

Osim njihove upotrebe u industriji, SCARA roboti se koriste i u edukativne svrhe. Njihova jednostavnija kinematika omogućuje lakše razumijevanje, dok njihova preciznost, točnost i fleksibilnost daju uvid u njihovu široku praktičnu primjenu u industriji. Većinom dolaze uz jednostavna korisnička sučelja čime olakšavaju upravljanje i programiranje samih robota.

Prilikom odabira robota koji bi se koristio u edukativne svrhe potrebno je obratiti pažnju na različite čimbenike. Odabrani robot trebao bi imati jednostavno sučelje koje je pristupačno i razumljivo za studente bez prethodnog iskustva u robotici. Potrebno je provjeriti dostupnost obrazovnih materijala poput različitih video materijala, dokumentacije i primjera programa koji bi olakšali učenje. Kako bi učenje bilo zanimljivije, fleksibilnost robota je jedan od ključnih faktora koji omogućava studentima upuštanje u različite zadatke i projekte. Iako industrijska preciznost nije glavni zahtjev, robot bi trebao osigurati dovoljno visoku preciznost i pouzdanost kako bi se osiguralo nesmetano, pozitivno i produktivno edukativno iskustvo. Kao i u industriji, sigurnost je najbitniji faktor. Prilikom interakcije s robotom potrebno je osigurati korisnika od mogućih ozljeda što je posebno važno tijekom učenja i eksperimentiranja s robotom. S obzirom na to da se robot koristi učestalo i u raznim primjenama potrebno osigurati izdržljivost i robusnost, a zbog svoje svrhe, robot bi trebao biti što manjih dimenzija bez potrebe za velikim nosivostima.

Nakon pregleda i usporedbe SCARA robota, u sklopu ovoga rada oblikovat će se edukativni SCARA robot, preuzet sa edukacijske web stranice How To Mechatronics [1], koji će se zatim sastaviti, testirati te programirati uz pomoć softverskog ROS paketa MoveIt.

2. PREGLED SCARA ROBOTA

Kako nije bilo dostupnih edukativnih SCARA robota, u ovome poglavlju napravit će se pregled industrijskih SCARA robota proizvođača ABB, Fanuc i Epson koji su svjetski priznati proizvođači robota. Prednosti odabira takvih proizvođača su već ustanovljena pouzdanost i kvaliteta njihovih proizvoda te zbog njihove dugogodišnje primjene u industriji, veća dostupnost obrazovnih resursa.

2.1. ABB IRB910SC

IRB 910SC je prva generacija SCARA robota proizvođača ABB Robotics. Nudi visoku preciznost i brze cikluse rada te sposobnost rada u ograničenim prostorima. Maksimalne nosivosti 6 kilograma i ponovljivosti 0,018 milimetara čine ovaj robot idealnim robotom za montažu i manipulaciju manjim dijelovima te inspekciju i kontrolu kvalitete. Ovaj ABB-ov robot može se programirati pomoću programskog jezika ABB RAPID te se kontrolirati pomoću softverskog paketa RobotWare koji omogućava potpunu kontrolu robotskog sustava. Također može se programirati i pomoću ABB-ovog alata za simuliranje i offline programiranje RobotStudio. Osim ABB-ovog softvera, ABB-ovi kontroleri mogu se kontrolirati pomoću ROS-a za koji postoje razvijeni ROS paketi sa svrhom olakšavanja interakcije između ABB robotskog kontrolera i ROS baziranih sustava. U tablici 2.1. nalaze se glavne karakteristike ovoga robota.

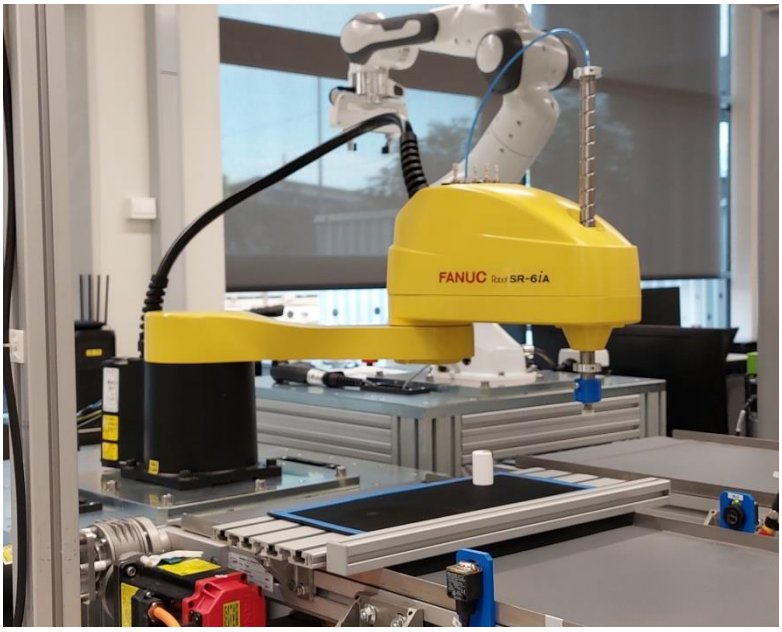
Slika 2.1. ABB IRB910SC [2]

Tablica 2.1. Karakteristike ABB IRB910SC robota

Nosivost	6 kg
Masa	25 kg
Doseg	550 mm
Ponovljivost	0,018 mm
Napajanje	200-600 V, 50/60Hz
Stupanj zaštite	IP20
Kontroler	IRC 5 Compact

2.2. Fanuc SR-6iA

Fanucov SR-6iA je kompaktan, brz i precizan SCARA robot. Za razliku od ABB-ovog IRB910SC robota, SR-6iA ima veći doseg te sličnu ponovljivost. Doseg ovog SCARA robota iznosi 650 milimetara, dok nosivost 6 kilograma. Primarne namjene ovoga SCARA manipulatora su montaža, „pick and place“, pakiranje te nanošenje raznih brtvila, ulja, ljepila i slično. Fanuc roboti se programiraju i upravljaju preko web baziranog softvera iRProgrammer, vrlo jednostavnog i intuitivnog korisničkog sučelja koje se na robot povezuje preko IP adrese. Fanuc također nudi razne edukativne programe.



Slika 2.2. CRTA - Fanuc SR-6iA

Tablica 2.2. Karakteristike Fanuc SR-6iA robota

Nosivost	6 kg
Masa	30 kg
Doseg	650 mm
Ponovljivost	0,01 mm
Napajanje	200-230V, 50/60Hz
Stupanj zaštite	IP20
Kontroler	R-30iB Compact Plus

2.3. Epson LS3-B SCARA

Epsonov LS3-B SCARA robot specifikacijama je sličan Fanucovom SR-3iA, istoga dosega, nosivosti i ponovljivosti. Kao i svi navedeni roboti, kompaktnog je dizajna, brz i precizan. Dolazi s ugrađenim priključkom za kameru što ga čini prikladnim robotom za široki spektar upotrebe. Epson je često orijentiran prema edukativnom tržištu i pruža podršku kroz razne edukativne materijale. Epson LS3-B SCARA može se programirati pomoću raznih alata i programskih jezika. Kao ABB i Fanuc, prvenstveno koriste vlastiti softverski paket za programiranje robota Epson RC+ 7.0, a pomoću RC+ 7.0 API-a roboti se mogu kontrolirati koristeći Visual Basic, Visual C++ i ostale programske jezike. Također, za izradu korisničkih sučelja može se koristiti Epsonov GUI Builder što dodatno olakšava učenje i programiranje Epsonovih robota. Sve informacije vezane uz robote i softver lako su dostupne.

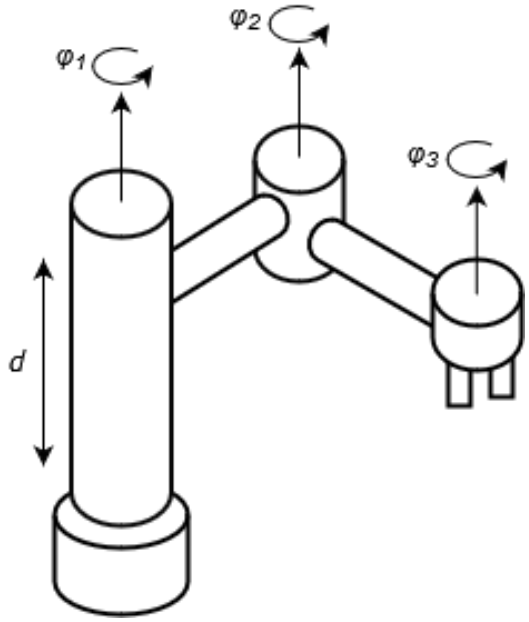
Slika 2.3. Epson LS3-B SCARA [4]

Tablica 2.3. Karakteristike Epson LS3-B SCARA robota

Nosivost	3 kg
Masa	14 kg
Doseg	400 mm
Ponovljivost	0,01 mm
Napajanje	200-230V, 50/60Hz
Stupanj zaštite	IP20
Kontroler	RC90 Controller

3. IZRADA EDUKATIVNOG SCARA ROBOTA

Konfiguracija SCARA manipulatora izrađenog u sklopu ovoga rada ima RTRR strukturu, odnosno 3 rotacijska i jedan translacijski zglob. Navedena struktura je odabrana kako bi se postigla stabilnost robota i povećao raspon translacijskog zgloba. SCARA manipulator sastoji se većinom od 3D tiskanih dijelova izrađenih pomoću 3D printera dostupnih u Regionalnom centru izvrsnosti za robotske tehnologije (CRTA), a dijelom od standardnih mehaničkih komponenata. U ovom poglavlju opisat će se redom 3D tiskanje dijelova, odabir standardnih mehaničkih komponenata te odabir aktuatorskog, senzorskog i upravljačkog sustava edukativnog SCARA robota.



Slika 3.1. Struktura SCARA manipulatora

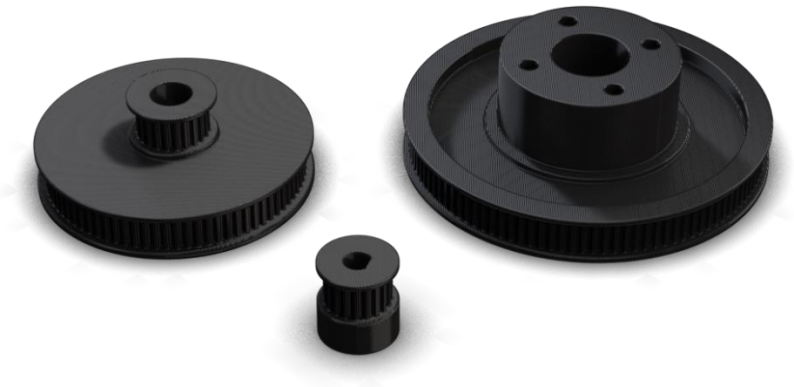
3.1. 3D tiskanje dijelova

Za izradu 3D tiskanih dijelova koristio se Prusa i3 MK3S 3D printer. Odabrani materijal je Prusament PETG koji je odličan odabir za tiskanje mehaničkih komponenti zbog visoke žilavosti, izdržljivosti te lakoće i brzine tiskanja. U tablici 3.1. nalazi se pregled svih dijelova koji su izrađeni postupkom 3D tiskanja, vrijeme potrebno za tiskanje na navedenom 3D printeru te utrošak materijala za izradu istih.

Tablica 3.1. Popis 3D tiskanih dijelova

RB.	NAZIV	CAD MODEL	VRIJEME IZRADE	UTROŠAK PETG
1.	Poklopac ruke 1	Arm 1 Cover	2 h i 56 min	44,7 g
2.	Ruka 1	Arm 1	10 h i 45 min	121,93 g
3.	Poklopac ruke 2	Arm 2 Cover	4 h i 27 min	58,38 g
4.	Ruka 2	Arm 2	9 h i 24 min	106,42 g
5.	Poklopac baze	Base cover	3 h i 7 min	47,13 g
6.	Baza	Base	20 h i 48 min	245,29 g
7.	Poklopac hvataljke	Gripper Cover	1h i 3 min	10,63 g
8.	Prst (x2)	Gripper End	1 h i 42 min	15,77 g
9.	Držač prsta (x4)	Gripper Hand	2 h i 14 min	29,3 g
10.	Poveznica prsta 1 (x2)	Gripper link 1	10 min	1,73 g
11.	Poveznica prsta 2 (x2)	Gripper link 2	17 min	2,05 g
12.	Klizač (x2)	Gripper Mechanism slider	2 h i 1 min	19,62 g
13.	Držač servo motora	Gripper Servo Holder	1 h i 22 min	12,87 g
14.	Hvataljka – zglob poveznica	Gripper to J3 connector	36 min	7,01 g
15.	GT2 pogonska remenica (x4)	GT2 Pulley – Parametric	49 min	6,63 g
16.	GT2 remenica 22-80 zubi	GT2 Pulley - 22 – 80 teeth	1 h i 2 min	11,25 g
17.	GT2 remenica 23-80 zubi	GT2 Pulley – 23 – 80 teeth	1 h i 4 min	11,76 g
18.	GT2 remenica 90 zubi	GT2 Pulley – 90 teeth – J3	1 h i 39 min	16,64 g
19.	GT2 remenica 92 zubi	GT2 Pulley – 92 teeth – J2	1 h i 35 min	17,01 g
20.	GT2 remenica 110 zubi	GT2 Pulley –110 teeth – J1	2 h i 5 min	23,77 g
21.	Zglob 1	J1 Coupler	5 h i 15 min	63,77 g
22.	Zglob 2	J2 Coupler	3 h i 1 min	34,67 g
23.	Zglob 3	J3 Coupler	2 h i 31 min	28,93 g
24.	Stezaljka vodilice (x8)	Smooth Rod Clamp	4 h i 38 min	44,92 g
25.	Poklopac	Top Cover	3 h i 33 min	50,13 g
26.	Dno z-osi	Z-axis Bottom Plate	3 h i 39 min	46,74 g
27.	Nosač z-osi	Z-axis Mount Platform	10 h i 30 min	117,16 g
28.	Vrh z-osi	Z-axis Top Plate	3 h i 45 min	46,01 g
UKUPNO			105 h i 58 min	1196,21 g

Za pripremu tiskanja dijelova korišten je Prusin softver PrusaSlicer. PrusaSlicer je softver koji služi za generiranje G-koda seciranjem željenog dijela u slojeve pomoću kojeg se upravlja 3D printer. Prilikom tiskanja nosivih dijelova, kao što su zglobovi i ruke, vrijednost perimetra povećana je na četiri radi postizanja veće čvrstoće istih, dok su ostali dijelovi tiskani zadanom vrijednošću dva. *Fill Density* je također ostavljen na zadanoj vrijednosti 15%. Ostali parametri tiskanja ostavljeni su na zadanim vrijednostima opcije tiskanja „0,2 mm QUALITY“ unutar softvera PrusaSlicer.



Slika 3.2. Modeli GT2 remenica za 3D tiskanje

3.2. Mehanički sustav SCARA robota

Prije odabira aktuatorskog, senzorskog i upravljačkog sustava, odabiru se ostale standardne mehaničke komponente SCARA robota. Rotacijsko gibanje postiže se remenim prijenosom 3D tiskanim GT2 remenicama s odgovarajućim GT2 remenima, dok se translacija rotacijskog gibanja aktuatora u linearno gibanje postiže pomoću navojnog vretena s maticom.

Slika 3.3. Translacije gibanja

Kako su točnost i preciznost ključni čimbenici SCARA robota, odabran je GT2 remenski prijenos. Također, otporan je na trošenje, što omogućava dug vijek trajanja te je vrlo jednostavan za montažu i podešavanje. Postoje razne konfiguracije ovih remena što omogućava fleksibilnost prilikom dizajniranja i konstruiranja.

Slika 3.4. GT2 remen

Za postizanje preciznog i stabilnog vertikalnog gibanja z osi SCARA robota koristi se trapezno navojno vreteno u kombinaciji s maticom. Navojna vretena su pouzdano rješenje zbog svoje visoke preciznosti i ponovljivosti pozicioniranja. Zbog svoje samokočnosti pružaju dodatnu sigurnost i pouzdanost u radu minimiziranjem nekontroliranih kretnji ili gubitaka pozicije. Kao i GT2 remenski prijenos, jednostavni su za integraciju i sposobna podnijeti velika opterećenja. Prijenos rotacijskog gibanja aktuatora na navojno vreteno postiže se aluminijskom spojnicom. Uz vreteno, z os SCARA robota podupiru i četiri linearne vodilice uparenih s linearnim ležajevima, dok su za podupiranje rotacijskih gibanja korišteni razni radijalni i aksijalni ležajevi. U tablici 3.2. nalazi se popis svih mehaničkih dijelova SCARA robota.

Tablica 3.2. Popis mehaničkih dijelova SCARA robota

RB.	NAZIV	BROJ KOMADA	NAPOMENA
1.	GT2 remen 200 mm	1	-
2.	GT2 remen 300 mm	2	-
3.	GT2 remen 400 mm	2	-
4.	Navojno vreteno s maticom	1	Ø8 mm, duljine 400 mm
5.	Vodilica	4	Ø10 mm, duljina 400 mm
6.	Elastična aluminijska spojnica	1	Ø5 mm na Ø8 mm
7.	Ležaj 608 2RS 8x22x7	5	-
8.	Ležaj 61806 2RS 30x42x7	3	-
9.	Ležaj 51107 35x52x12	2	-
10.	Ležaj 51108 40x60x13	1	-
11.	Ležaj 51107 35x52x12	2	-
12.	Ležaj 51108 40x60x13	1	-
13.	Ležaj 61807 2RS 35x47x7	1	-
14.	M3x10	4	-
15.	M3x12	6	-
16.	M3x14	12	-
17.	M3x16	2	-
18.	M3x20	8	-
19.	M3x30	10	-
20.	M4x16	4	-
21.	M4x20	14	-
22.	M4x25	24	-
23.	M4x35	8	-
24.	M4x50	8	-
25.	M5x20	4	-
26.	M8x45	2	-

3.3. Podsustavi SCARA robota

SCARA robot sastoji se od četiri koračnih motora uparenih s četiri mikro prekidača, servomotora i mikro kontrolera. Koračni motori koriste se za provedbu rotacijskih i translacijskih gibanja, dok se servomotor koristi za upravljanje prstima ruke. Cijeli sustav i robot upravljaju se pomoću mikro kontrolera s odgovarajućim štitom i upravljačkim jedinicama.

3.3.1. Aktuatorski i senzorski sustav

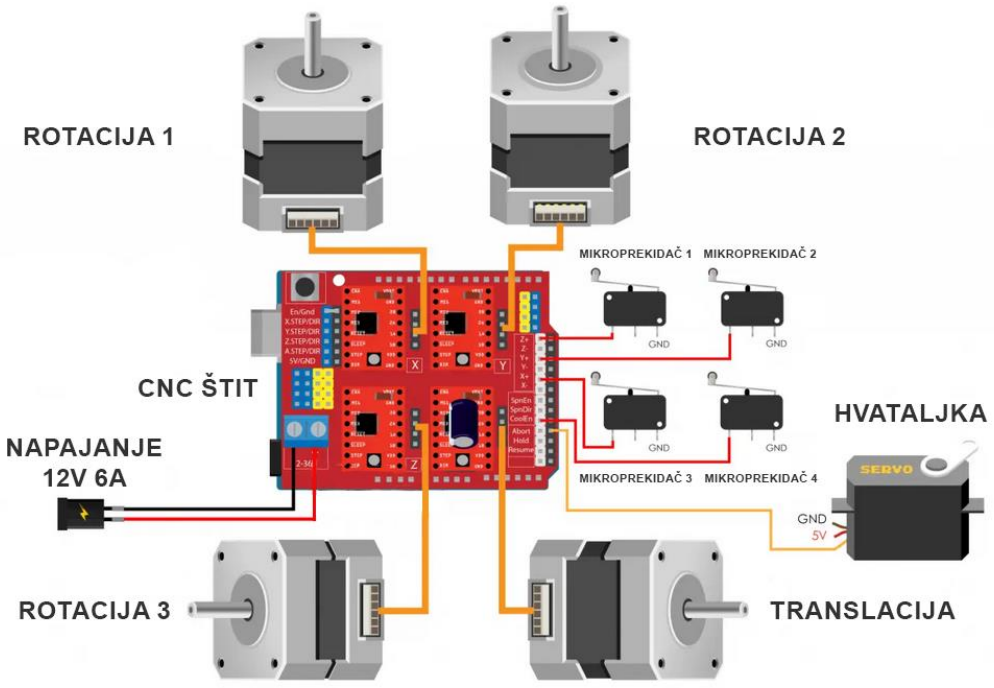
Koračni motori odličan su odabir za postizanje preciznih gibanja bez potrebe za enkoderom. Također, imaju visok prijenos snage i momenta s obzirom na svoje dimenzije. NEMA 17 koračni motori koriste se u raznim 3D printerima i manjim CNC glodalicama. Kontroliraju se otvorenom petljom što čini sustav manje osjetljivim na smetnje i lakšim za održavanje. Koračni motori zajedno s odgovarajućim prijenosnicima i mehaničkim komponentama omogućuju postizanje složenih kretanja s visokom preciznošću. Na vratila koračnih motora koja se koriste za pogon rotacijskih gibanja ugrađena je GT2 remenica koja prenosi okretni moment na zglobove SCARA robota, dok za pogon translacijskih gibanja vratilo je upareno s navojnim vretenom pomoću aluminijske spojnice. Kako bi se saznao položaj koračnih motora, upareni su s mikro prekidačima koji omogućuju određivanje početnog položaja SCARA robota, dok je svaki sljedeći položaj poznat putem praćenja broja koraka koje motor izvršava, a koji se zadaje pomoću mikrokontrolera. Zbog svojih kompaktnih dimenzija i male mase, za kontrolu hvataljke, odabran je servomotor MG996R.

Slika 3.5. NEMA 17 koračni motor s GT2 remenicom

3.3.2. Upravljački sustav

Sustav upravljanja SCARA robota sastoji se od ploče mikrokontrolera s odgovarajućim štitom i driverima. Za upravljanje SCARA robotom obrađenim u ovome radu odabran je Arduino Mega 2560 Rev V3 ploča mikro kontrolera bazirana na ATmega2560 s velikom brzinom procesiranja. Otvorenog je koda te je opremljena skupovima digitalnih i analognih ulazno/izlaznih pinova pomoću koji se procesiraju i šalju signali. Kako bi se ostvarila kontrola i napajanje koračnih motora i servomotora te procesiranje signala mikro prekidača koristi se CNC štit V3.

CNC štit omogućava upravljanje do 4 koračnih motora te se koristi za kontrolu kretanja u raznim aplikacijama kao što su 3D printeri i CNC strojevi. Kompatibilan je sa A4988 ili DRV8825 driverima za kontrolu koračnih motora koji omogućavaju podešavanje mikro koraka za precizniju kontrolu kretanja te podržavaju široki raspon za napajanje motora što omogućava veću fleksibilnost pri odabiru napajanja za željenu primjenu. Na slici 3.6. vidljiv je dijagram spajanja CNC štita s električnim komponentama, dok se u tablici 3.3. nalazi popis svih komponenata elemenata elektroničkog sustava upravljanja.



Slika 3.6. Dijagram spajanja CNC štita [1]

Tablica 3.3. Popis elemenata elektroničkog sustava upravljanja

RB.	NAZIV	BROJ KOMADA
1.	MG996R Servo motor	1
2.	NEMA 17 koračni motor	4
3.	Arduino Mega 2560 Rev V3	1
4.	Arduino CNC štit	1
5.	A4988 driver	4
6.	DC Napajanje Mean Well LRS-100-12	1
7.	Mikro prekidač	4

Slika 3.7. 3D model SCARA robota

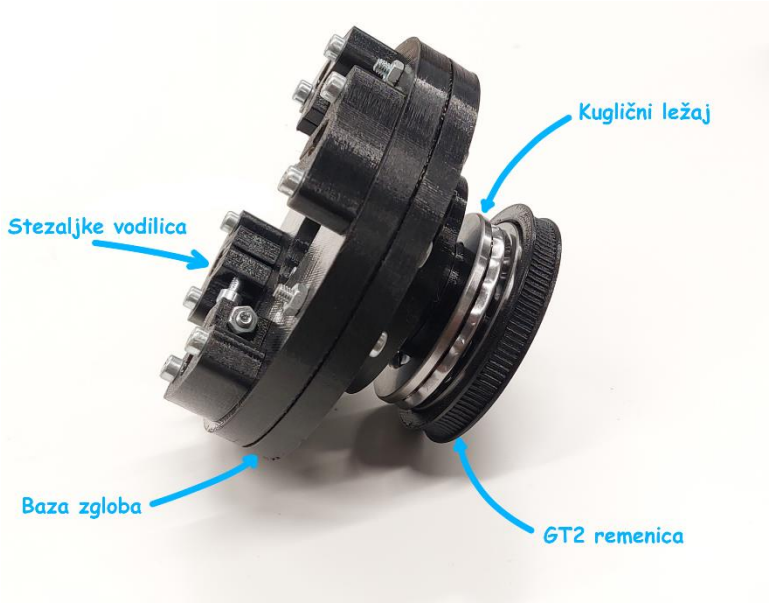
3.4. Sastavljanje SCARA robota

Nakon 3D tiskanja dijelova, odabira standardnih mehaničkih komponenta te elektroničkog sustava upravljanja robotom, SCARA robot se može sastaviti. Postavljaju se radijalni ležajevi u zato predviđena mjesta za omogućavanje rotacije GT2 remenica.

Slika 3.8. GT2 remenski prijenos

Slika 3.9. GT2 remenica s kugličnim ležajem

Svi rotacijski zglobovi robota izvedeni na isti način. Krajnja pogonska remenica povezana je sa kugličnim ležajem naslonjenim na utisnuti radijalni ležaj. Na drugoj strani zgloba nalazi se poveznica zgloba koja je vijčano spojena na istu remenicu.

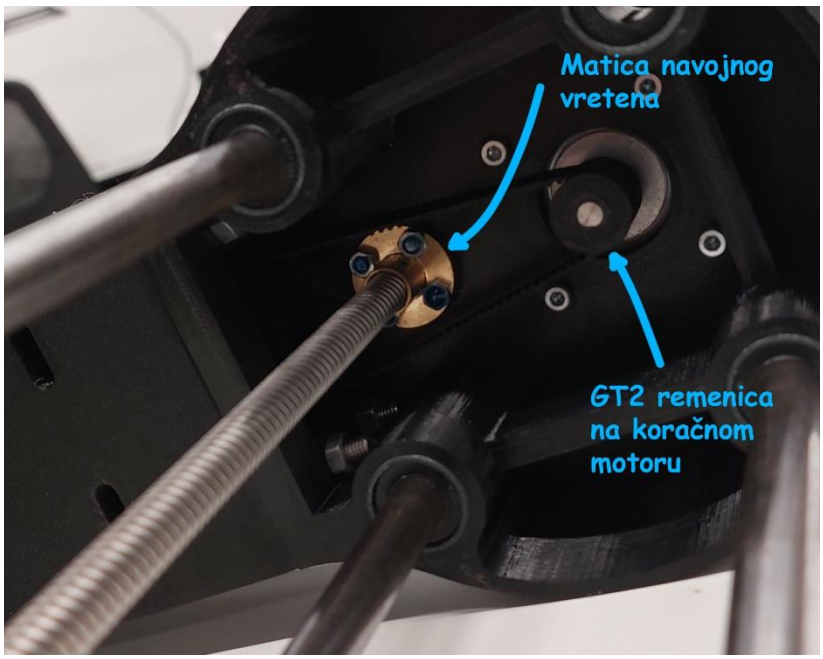


Slika 3.10. Prvi rotacijski zglob

Zatim se sklapa robotska ruka koja se izvodi na sličan način kao i bazni segment robota. Robotska ruka sastoji se od 2 rotacijska zgloba te se na njoj nalazi i matica navojnog vretena pomoću koje se provodi translacija robota po z osi.

Slika 3.11. Robotska ruka

Translaciju robota po z osi također podržavaju četiri vodilice uparene s linearnim ležajevima.

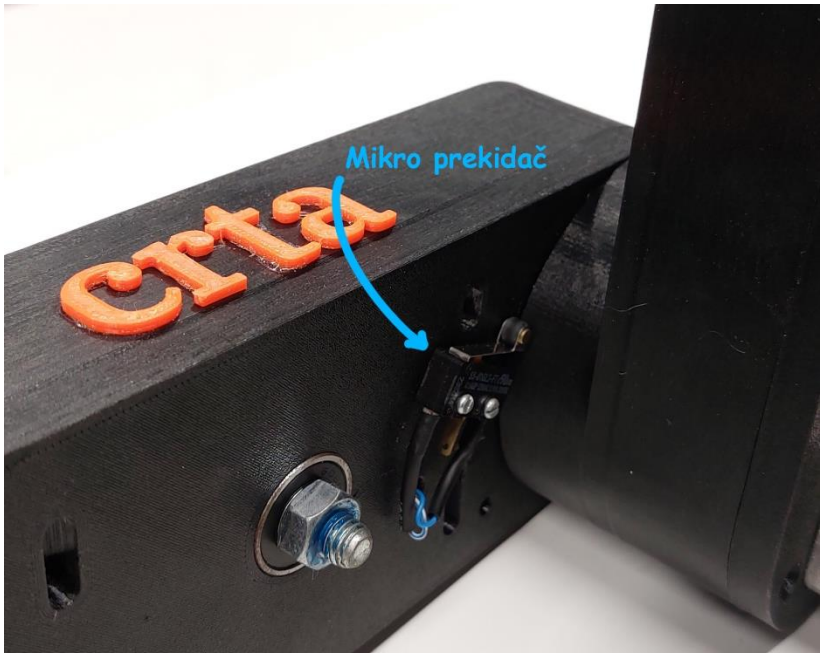


Slika 3.12. Matica navojnog vretena

Nakon sklapanja robotske ruke provode se vodiči za pogon NEMA 17 koračnih motora te signale mikro prekidača. Na dijelovima ruke istiskani su držači vodiča kako ne bi smetali prilikom kretnji robota.

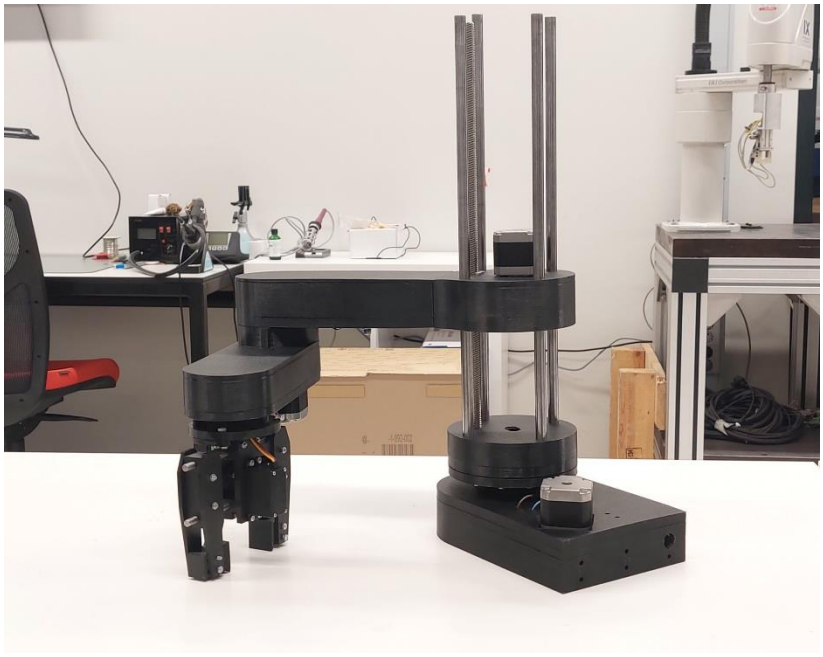
Slika 3.13. Ožičenje ruke SCARA robota

Kut rotacije pojedinog zgloba ograničen je pomoću mikro prekidača pomoću kojih se dobiva početni položaj pri upravljanju robotom.



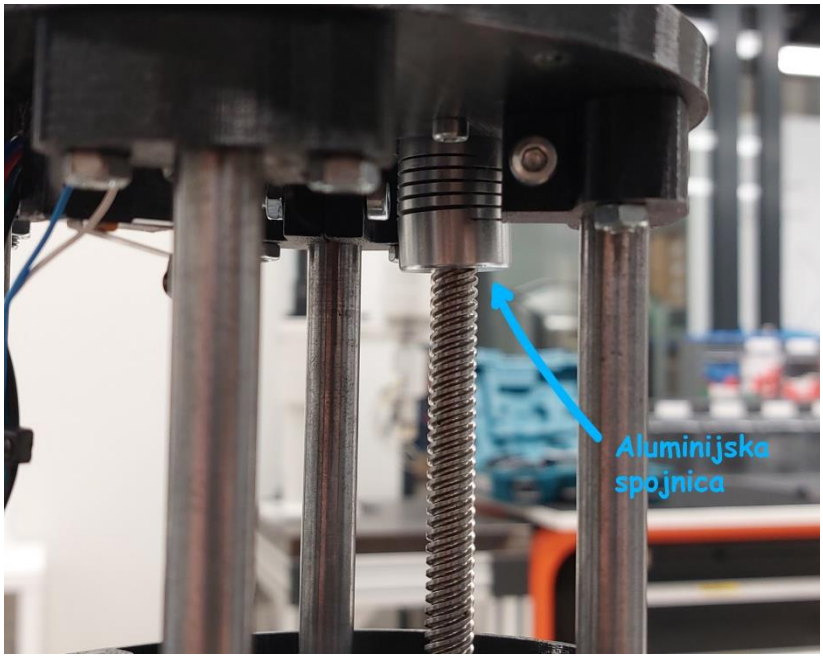
Slika 3.14. Mikro prekidač

Ruka se zatim povezuje s prvim zglobom pomoću stezaljki vodilica koje pridržavaju cijelog robota.



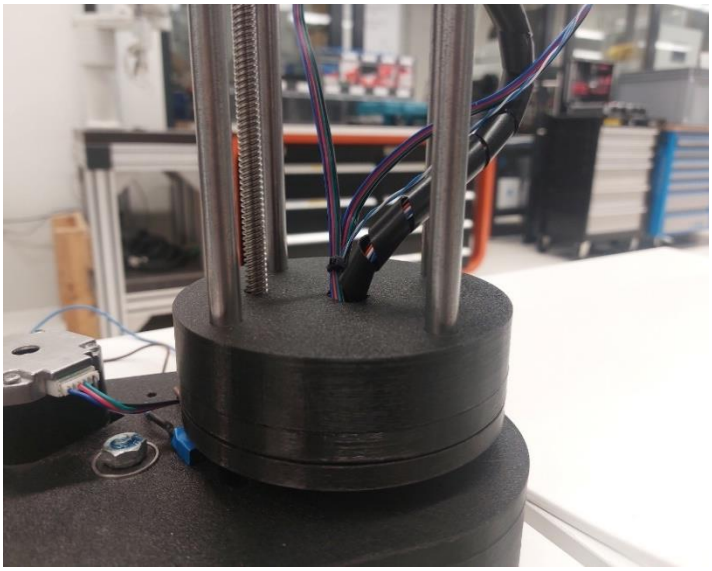
Slika 3.15. Robotska ruka

Navojno vreteno povezano je sa koračnim motorom pomoću aluminijske spojnice te tako prenosi rotacijsko gibanje aktuatora u translacijsko gibanje po z osi SCARA robota. Kao i na dnu translacijskog zgloba, na vrhu se također nalaze stezaljke koje učvršćuju strukturu robota. Uz koračni motor nalazi se mikro prekidač.



Slika 3.16. Aluminijska spojnica

U originalnom dizajnu robota organizacija vodiča je loše odrađena. Pri rotaciji prvoga zgloba kablovi bi se rotirali zajedno sa robotom te bi došlo do problema petljanja istih te povlačenja Arduina i napajanja. Problem je riješen bušenjem rupe na dnu translacijskog zgloba i u bazi robota te provlačenjem kablova kroz njih. Na taj način kablovi su organiziraniji te nikakva rotacija ne utječe na njih.



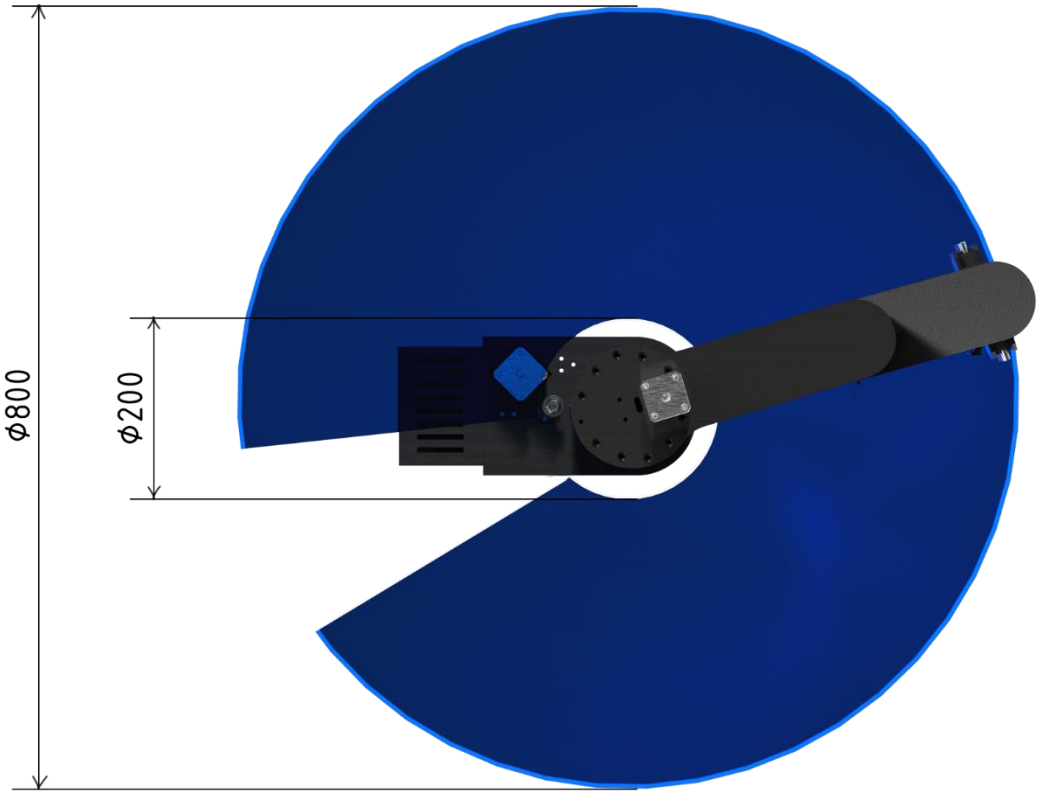
Slika 3.17. Organizacija kablova

Robot također ima problema sa stabilnošću koji bi se mogli riješiti povećanjem baze, skraćivanjem segmenata qruke, korištenjem manjih koračnih motora ili većim ležajevima prvoga zgloba. Također, pri idućim tiskanjima preporuča se povećanje *Fill density.* Kablovi se spajaju na Arduino prema dijagramu danom na slici 3.6. Konačni robot ima doseg od 400 milimetara po *x, y* ravnini te 250 milimetara po *z* osi. Koristi napajanje od 12V, 6A. Brzine zglobova se podešavaju unutar Arduino koda te su postavljene prema tablici 3.4.

Slika 3.18. SCARA robot

Tablica 3.4. Brzine zglobova

RB.	NAZIV ZGLOBA	BRZINA (broj koraka po sekundi)
1.	Rotacijski zglob baze	1500
2.	Translacijski zglob	2000
3.	Rotacijski zglob robotske ruke	2000
4.	Rotacijski zglob hvataljke	2000



Slika 3.19. Radni prostor

4. KINEMATIKA SCARA ROBOTA

Kinetika proučava kretanje tijela bez razmatranja sila ili momenata koji uzrokuju to kretanje. Robotička kinematika odnosi se na analitičko proučavanje kretanja robotskog manipulatora. Razvoj odgovarajućih kinematičkih modela za mehanizam robota ključan je za analizu ponašanja industrijskih manipulatora. U kinematičkom modeliranju manipulatora obično se koriste dvije osnovne prostorne kategorije: kartezijanski prostor i kvaternionski prostor. Prijelaz između dvaju kartezijanskih koordinatnih sustava može se razložiti na rotaciju i translaciju. Postoji različiti načini za prikazivanje rotacije, uključujući Eulerove kutove, Gibbsov vektor, Cayley-Klein parametre, Paulijeve matrice vrtnje, os i kut, ortonormirane matrice i Hamiltonove kvaternione. Od tih prikaza, homogene transformacije temeljene na 4x4 matricama (ortonormiranim matricama) najčešće se koriste u robotici. Denavit i Hartenberg pokazali su da opća transformacija između dva segmenta zahtijeva svega četiri parametra.

Kinematika robota može se podijeliti u direktnu kinematiku i inverznu kinematiku. Direktna kinematika predstavlja relativno jednostavan problem s jednadžbama koje se lako izvode, što osigurava dostupnost rješenja za položaj i orijentaciju manipulatora. S druge strane, inverzna kinematika predstavlja znatno složeniji izazov u usporedbi s direktnom kinematikom. Rješavanje problema inverzne kinematike zahtijeva znatno računalno opterećenje i obično zahtijeva značajno vrijeme, posebno kada se primjenjuje u stvarnom vremenu za upravljanje manipulatorima.

4.1. Direktna kinematika

Roboti se sastoje od niza serijskih veza koje su povezani rotacijskim ili translacijskih zglobova od baze robota sve do alata. Direktnom kinematikom izračunava se položaj i orijentacija alata na temelju varijabli zglobova. Denavit-Hartenbergova metoda najčešća je metoda za opisivanje kinematike robota. Prema njoj zapis strukture robota glasi:

1. Os z_{i-1} koordinatnog sustava, leži u osi gibanja i-tog stupnja gibanja.
2. Os x_{i-1} okomita je na z_{i-1} i paralelna je s osi koja ide uzduž segmenta.
3. Os y_{i-1} postavlja se tako da čini desnokretni koordinatni sustav.

Pomoću zapisa strukture robota, položaj i orijentacija koordinatnih sustava su u potpunosti definirani preko idućih četiri Denavit-Hartenbergovih parametara:

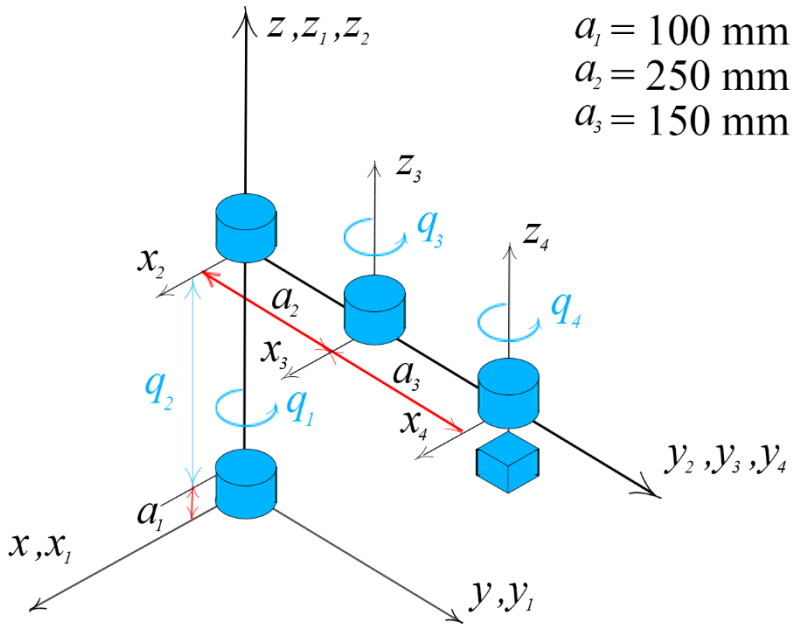
a_i – duljina segmenta, odnosno duljina okomice između z_{i-1} i z_i

d_i – pomak segmenta, odnosno duljina duž z_i osi između sjecišta z_i s a_{i-1} i z_i s a_i

α_{i-1} – kut uvijanja, odnosno kut između z_{i-1}i z_i oko osi x_{i-1}

θ_i – kut zgloba, odnosno kut između x_{i-1} i x_i oko osi z_i.

SCARA roboti imaju paralelne z osi, pa iz tog razloga parametar α_i u svim matricama homogenih transformacija poprima vrijednost 0, dok parametar a_i ovisi o geometriji manipulatora. Ostali parametri su varijabilni. Na slici 4.1. prikazan je SCARA manipulator s njegovim stupnjevima slobode.



Slika 4.1. Kinematska struktura SCARA manipulatora

4.1.1. Matrice homogenih transformacija

Prije rješavanja kinematičkog problema, potrebno je definirati izraze koji se koriste za transformaciju m-tog koordinatnog sustava u n-ti. Za transformaciju translacije m-tog koordinatnog sustava u odnosu na n-ti koordinatni sustav u smjerovima x, y i z osi gdje su udaljenosti definirane vrijednostima a, b i c glasi (4.1):

$$^{\mathbf{m}}\mathbf{A_n} = \mathrm{Tran}(a,b,c) = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad \textbf{(4.1)}$$

Dok je izraz za transformaciju rotacije oko z osi za kut α (4.2):

$$^{\mathbf{m}}\mathbf{A_n} = \mathrm{Rot}(z,\alpha) = \begin{bmatrix} cos\alpha & -sin\alpha & 0 & 0 \\ sin\alpha & cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad \textbf{(4.2)}$$

Radi pojednostavljenja prikaza proračuna uvode se sljedeće supstitucije (4.3):

$$\begin{aligned} s_i &= sin(q_i), \\ c_i &= cos(q_i), \end{aligned} \quad \textbf{(4.3)}$$

gdje q_i predstavlja kut rotacije oko z osi.

Matrica homogenih transformacija prvog segmenta sastoji se od rotacije koordinatnog sustava u odnosu na početni te glasi (4.4):

$$^{\mathbf{0}}\mathbf{A_1} = \mathrm{Rot}(z,q_1) = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad \textbf{(4.4)}$$

Drugi zglob SCARA robota pomaknut je u odnosu na *z* os prvoga koordinatnog sustava za fiksnu vrijednost a_1 te varijabilnu vrijednost translacije q_2, pa matrica homogenih transformacija drugog segmenta glasi (4.5):

$$^{\mathbf{1}}\mathbf{A_2} = \mathrm{Tran}(0\ ,0\ ,a_1 + q_2\ ) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & a_1 + q_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad \textbf{(4.5)}$$

Treći zglob je rotacijski zglob oko *z* osi te zamaknut je u odnosu na *y* os koordinatnog sustava prethodnog zgloba za fiksnu vrijednost a_2. Matrica homogenih transformacija onda glasi (4.6):

$$^{2}\mathbf{A_3} = \mathrm{Tran}(0, a_2, 0)\mathrm{Rot}(z, q_3) = \begin{bmatrix} c_3 & -s_3 & 0 & 0 \\ s_3 & c_3 & 0 & a_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad \textbf{(4.6)}$$

Finalni zglob je također rotacijski zglob oko z osi zamaknut u odnosu na x os koordinatnog sustava prethodnog zgloba za fiksnu vrijednost a_3. Matrica homogenih transformacija onda glasi (4.7):

$$^{3}\mathbf{A_4} = \mathrm{Tran}(0, a_3, 0)\mathrm{Rot}(z, q_4) = \begin{bmatrix} c_4 & -s_4 & 0 & 0 \\ s_4 & c_4 & 0 & a_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad \textbf{(4.7)}$$

Direktna transformacija položaja i orijentacije vrha alata u odnosu na početni, nepomični koordinatni sustav, definira se izrazom (4.8):

$$^{0}\mathbf{T_4} = {}^{0}\mathbf{A_1}\ {}^{1}\mathbf{A_2}\ {}^{2}\mathbf{A_3}\ {}^{3}\mathbf{A_4}, \quad \textbf{(4.8)}$$

Te se rješavanjem sustava dobiva izraz (4.9):

$$^{0}\mathbf{T_4} = \begin{bmatrix} c_{134} & -s_{134} & 0 & a_2c_1 + a_3c_{13} \\ s_{134} & c_{134} & 0 & a_2s_1 + a_3s_{13} \\ 0 & 0 & 1 & a_2 + q_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad \textbf{(4.9)}$$

Izraz (4.9)(4.8) može se zapisati u obliku (4.10):

$$^{0}\mathbf{T_4} = \begin{bmatrix} \mathbf{n} & \mathbf{o} & \mathbf{a} & \mathbf{p} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(4.10)}$$

gdje vektori **n**, **o**, **a** i **p** predstavljaju vektore normale, orijentacije, djelovanja i položaja. Definira se i vektor unutarnjih koordinata, **q**, koji predstavlja stupnjeve slobode gibanja odnosno rotacije i translacije koje vrši robot koji glasi (4.11):

$$\mathbf{q} = [q_1 \quad q_2 \quad q_3 \quad q_4]^{\mathrm{T}} \tag{4.11}$$

Te se definira vektor vanjskih koordinata, **r**, koji se sastoji od kartezijevih koordinata, koje definiraju položaj, i Eulerovih kutova, koji definiraju orijentaciju. Zglobovi SCARA robota ograničeni su na rotaciju oko vertikalne osi, pa su kut posrtanja φ i kut valjanja ψ jednaki 0. Kut skretanja SCARA robota glasi (4.12):

$$\vartheta = q_1 + q_3 + q_4 \tag{4.12}$$

Konačni izraz vektora vanjskih koordinata SCARA robota glasi:

$$\mathbf{r} = [p_x \quad p_y \quad p_z \quad \vartheta]^{\mathrm{T}} \tag{4.13}$$

te iz izraza (4.9) i (4.10) slijede izrazi (4.14), (4.15) i (4.16):

$$p_x = a_2 c_1 + a_3 c_{13} \tag{4.14}$$

$$p_y = a_2 s_1 + a_3 s_{13} \tag{4.15}$$

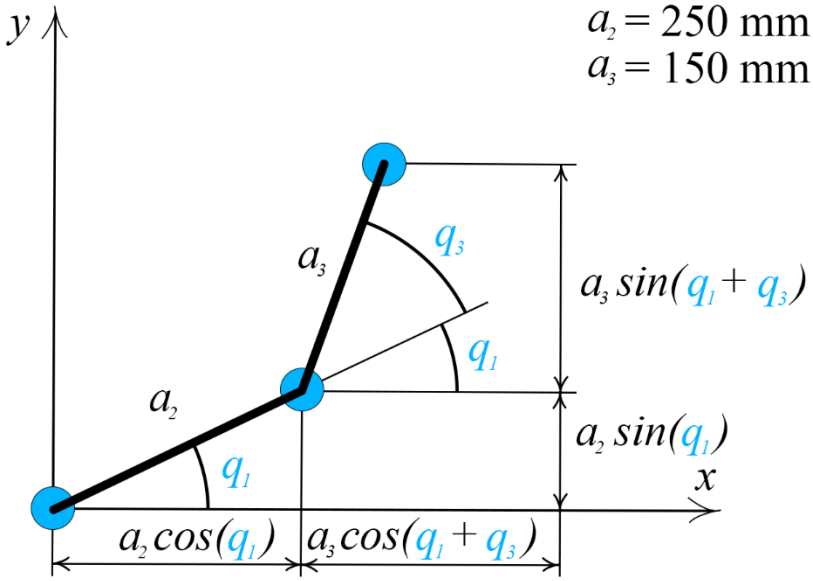
$$p_z = a_1 + q_2 \tag{4.16}$$

4.2. Inverzna kinematika

Za razliku od direktne kinematike, problem inverzne kinematike je pronaći rotacije i translacije zglobova preko zadanih kartezijevih koordinata (položaja i orijentacije alata), odnosno transformaciju vanjskih koordinata manipulatora u unutarnje. Problem inverzne kinematike je nepostojanje univerzalnog algoritma za rješavanje problema i postojanje više rješenja od kojih sva nisu nužno konstrukcijski izvediva. Ako se željena pozicija i orijentacija nalazi unutar radnoga prostora manipulatora mora postojati barem jedno rješenje problema. Iz izraza (4.16) može se odmah zaključiti rješenje translacijskog zgloba koje glasi:

$$q_2 = p_z - a_1 \tag{4.17}$$

Kinematika SCARA robota je pojednostavljena s 2D problemom, gdje je na kraju kinematskog lanca dodan još jedan stupanj slobode gibanja, zadužen za orijentaciju alata.



Slika 4.2. 2D pojednostavljenje problema

Ponavljaju se izrazi (4.14) i (4.15) radi preglednosti:

$$p_x = a_2 c_1 + a_3 c_{13} \tag{4.18}$$

$$p_y = a_2 s_1 + a_3 s_{13} \tag{4.19}$$

Sumom kvadriranih jednadžbi (4.18) i (4.19) izvodi se rješenje za prvi rotacijski stupanj slobode gibanja (4.20):

$$p_x{}^2 + p_y{}^2 = a_2{}^2 c_1{}^2 + a_3{}^2 c_{13}{}^2 + 2a_2 a_3 c_1 c_{13} + a_2{}^2 s_1{}^2 + a_3{}^2 s_{13}{}^2 + 2a_2 a_3 s_1 s_{13} \tag{4.20}$$

Uvrštavanjem $(c_i{}^2 + s_i{}^2) = 1$ i sređivanjem izraza slijedi izraz (4.21):

$$p_x{}^2 + p_y{}^2 = a_2{}^2 + a_3{}^2 + 2a_2 a_3 c_3 \tag{4.21}$$

Iz čega slijedi finalni izraz trećeg stupnja slobode gibanja (4.22):

$$q_3 = \cos^{-1}\left(\frac{p_x{}^2 + p_y{}^2 - a_2{}^2 - a_3{}^2}{2a_2 a_3}\right). \tag{4.22}$$

Pomoću istih parametra, potrebno je izračunati i prvi stupanj slobode gibanja. Jednadžba (4.18) množi se sa c_1,a jednadžba (4.19) sa s_1, pa slijede sljedeći izrazi:

$$c_1 p_x = a_2 {c_1}^2 + a_3 {c_1}^2 c_3 - a_3 c_1 s_1 s_3 \quad \textbf{(4.23)}$$

$$s_1 p_y = a_2 {s_1}^2 + a_3 {s_1}^2 c_3 + a_3 s_1 c_1 s_3 \quad \textbf{(4.24)}$$

Izrazi (4.23) i (4.24) se zbrajaju, pa slijedi:

$$c_1 p_x + s_1 p_y = a_2({c_1}^2 + {s_1}^2) + a_3 c_3({c_1}^2 + {s_1}^2), \quad \textbf{(4.25)}$$

koji pojednostavljivanjem glasi:

$$c_1 p_x + s_1 p_y = a_2 + a_3 c_3. \quad \textbf{(4.26)}$$

Jednadžba (4.18) množi se sa $-s_1$,a jednadžba (4.19) sa c_1, pa slijede sljedeći izrazi:

$$-s_1 p_x = -a_2 - s_1 c_1 + a_3 s_1 c_1 c_3 + a_3 {s_1}^2 s_3 \quad \textbf{(4.27)}$$

$$c_1 p_y = a_2 c_1 s_1 + a_3 c_1 s_1 c_3 + a_3 {c_1}^2 s_3, \quad \textbf{(4.28)}$$

Koji se sumiraju te pojednostavljuju:

$$-s_1 p_x + c_1 p_y = a_3 s_3. \quad \textbf{(4.29)}$$

Izraz (4.26) množi se sa p_x, a izraz (4.27) sa p_y te se također zbrajaju, pa sređivanjem slijedi izraz:

$$c_1 = \frac{p_x(a_2 + a_3 c_2) + p_y a_3 s_2}{{p_x}^2 + {p_y}^2}. \quad \textbf{(4.30)}$$

Dok se s_1 izračuna pravilom $({c_i}^2 + {s_i}^2) = 1$ te glasi:

$$s_1 = \pm\sqrt{1 - \left(\frac{p_x(a_2 + a_3 c_2) + p_y a_3 s_2}{{p_x}^2 + {p_y}^2}\right)^2}. \quad \textbf{(4.31)}$$

Kao rezultat, za prvi stupanj slobode gibanja postoje dva rješenja koja glase:

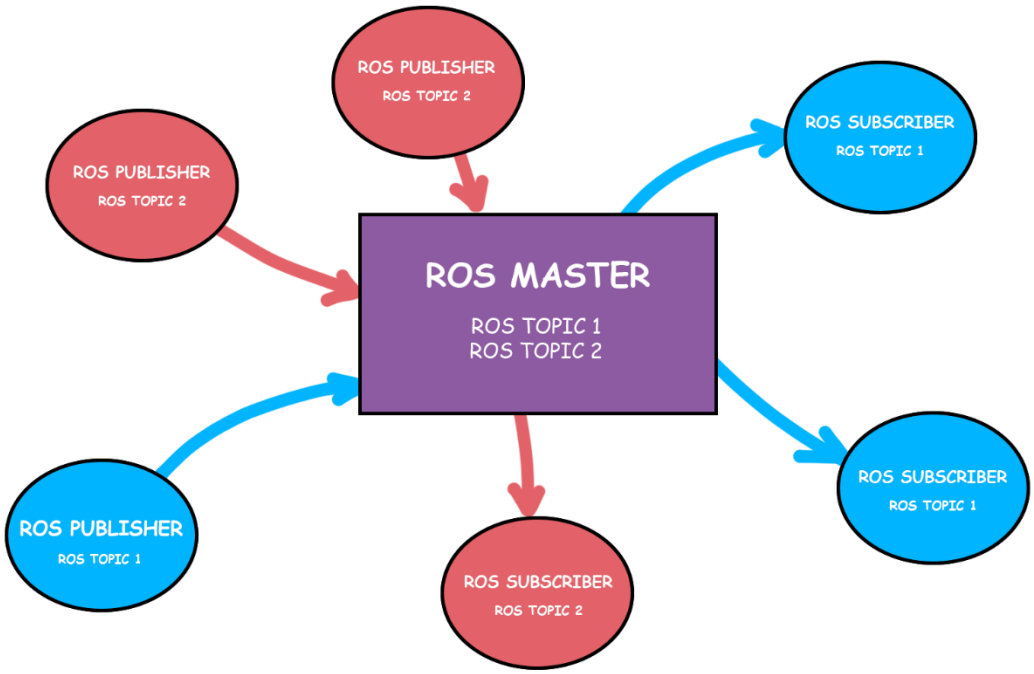
$$q_1 = Atan2\left(\pm\sqrt{1 - \left(\frac{p_x(a_2 + a_3 c_2) + p_y a_3 s_2}{{p_x}^2 + {p_y}^2}\right)^2}, \frac{p_x(a_2 + a_3 c_2) + p_y a_3 s_2}{{p_x}^2 + {p_y}^2}\right). \quad \textbf{(4.32)}$$

Finalni stupanj slobode gibanja računa se pomoću izraza (4.12):

$$q_4 = q_1 + q_3 - \vartheta \quad \textbf{(4.33)}$$

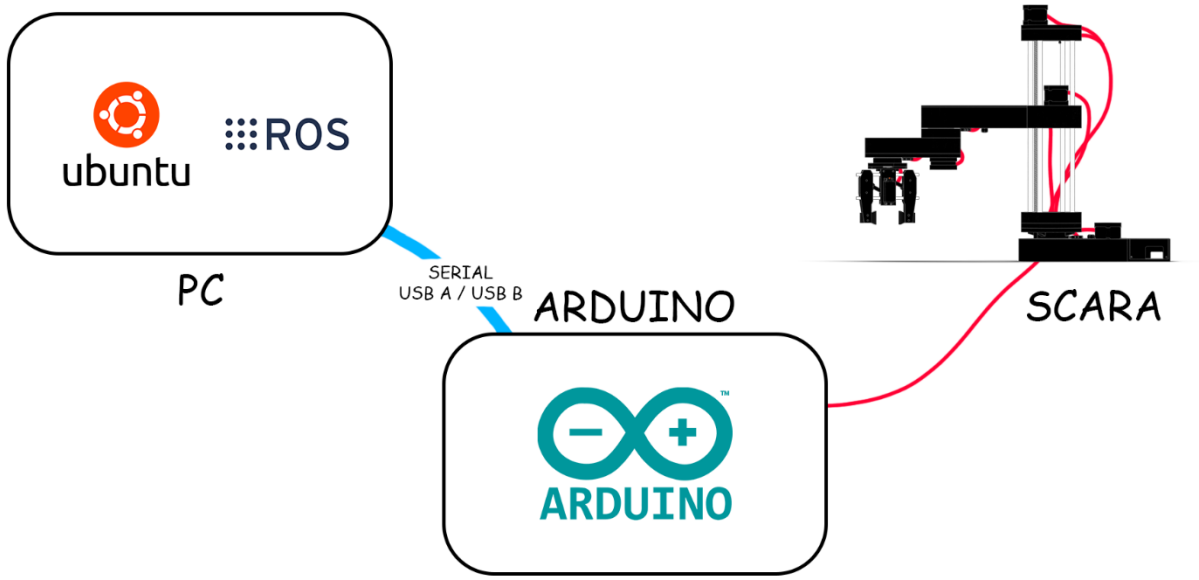
5. ROS

The Robot Operating System (ROS) je besplatno radno okruženje otvorenog koda koje služi za razvoj i kontrolu robota i robotskih aplikacija. Kombinacija je skupova upravljačkih algoritama i programa te alata za vizualizaciju. Cilj razvoja ROS-a je pojednostavljenje i približavanje robotike svima, od stručnjaka do hobista. Podržava više programskih jezika kao što su C++, Python, Octave i LISP. Kako bi se smanjila kompleksnost ROS-a, koristi se razni alati koji imaju specifičnu svrhu unutar ROS okruženja. Osnovni princip rada ROS-a je stvaranje raznih tema preko kojih čvorovi razmjenjuju informacije. Svaki od tih čvorova ima specifičnu funkciju te može imati ulogu „pošiljatelja“ i „pretplatnika“. Pošiljatelji šalju poruke na određenu ROS teme koje primaju i procesiraju samo pretplatnici na tu specifičnu temu. Tako se izgrađuje mreža jednostavnijih čvorova na kojima je lakše pronalaženje grešaka te otklanjanje istih. ROS Master zadužen je za održavanje komunikacije i pronalaženje čvorova.



Slika 5.1. Pojednostavljen prikaz načina rada ROS-a

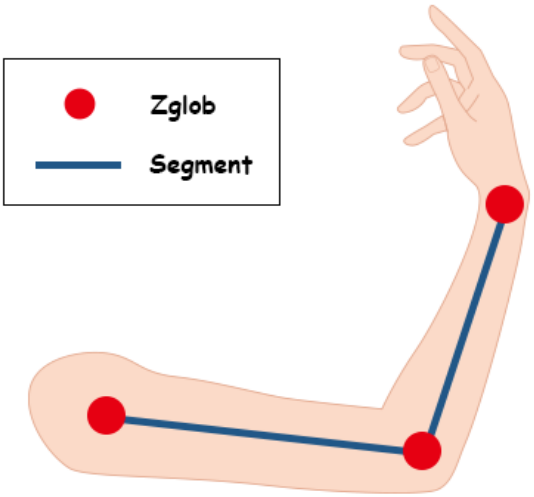
U ovome radu koristi se ROS *noetic*, dizajniranom za Ubuntu 20.04, ROS-ov softverski paket MoveIt i Python3. Postupak instalacije ROS-a i MoveIt-a detaljno je opisan na službenim stranicama ROS-a.



Slika 5.2. Shema komunikacije

5.1. URDF

Unified Robot Description Format (URDF) predstavlja XML-format koji se sve više koristi u području robotike, posebno u okviru ROS-a, za precizno opisivanje karakteristika i konfiguracije robota. Koristi se za simulaciju, planiranje kretanja, vizualizaciju i kontrolu robota u različitim robotskim okruženjima. Pomoću URDF datoteka definiraju se zglobovi, segmenti, senzori i geometrija robota. Kao i svaka XML datoteka, sadrži razne XML elemente koji su složeni u strukturu XML stabla.



Slika 5.3. URDF primjer [13]

URDF datoteka započinje <*robot*> XML elementom unutar kojeg se definiraju <*link*> elementi sa svojim inercijskim i vizualnim svojstvima te <*joint*> elementi koji opisuju kinematiku i dinamiku zglobova te segmente koje povezuju. Primjeri <*link*> i <*joint*> elemenata dani su u sljedećim isječcima koda *gazebo_scara.urdf* datoteke korištene u sklopu ovoga rada.

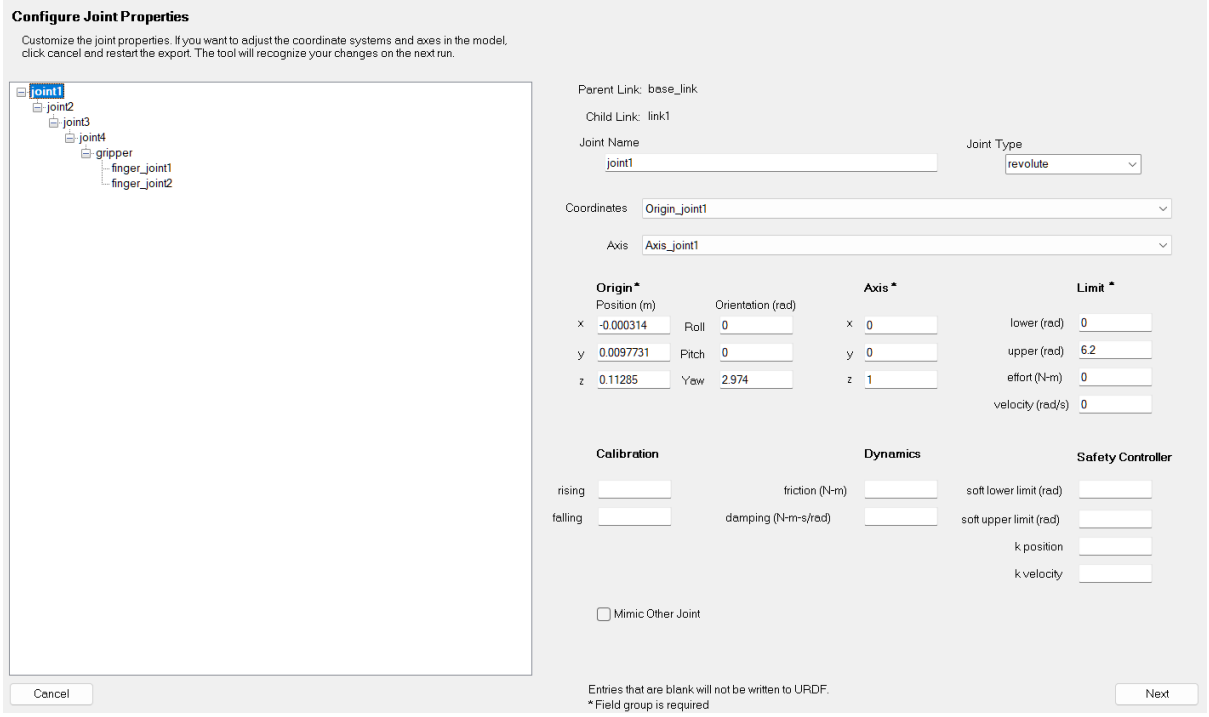
Primjer <*link*> elementa baznog segmenta SCARA robota:

```
<link name="base_link">
      <inertial>
            <origin xyz="-0.0559432 0.10740 -0.01367045" rpy="0 0 0"/>
            <mass value="0.56769603"/>
            <inertia ixx="0.001178" ixy="-1.3272E-05" ixz="-5.74547E-06"
iyy="0.00333381" iyz="-5.819175" izz="0.00233"/>
      </inertial>
      <visual>
            <origin xyz="0 0 0" rpy="0 0 0" />
            <geometry>
                <mesh filename="package://scara/meshes/base_link.STL"/>
            </geometry>
            <material name="">
                <color rgba="0.792156 0.819607843 0.93333 1"/>
            </material>
       </visual>
       <collision>
            <origin xyz="0 0 0" rpy="0 0 0"/>
            <geometry>
                <mesh filename="package://scara/meshes/base_link.STL"/>
            </geometry>
       </collision>
</link>
```

Primjer <*joint*> elementa *revolute* zgloba između baznog i prvog segmenta SCARA robota:

```
<joint name="joint1" type="revolute">
      <origin xyz="-0.000314 0.0097731 0.11285" rpy="0 0 2.974"/>
      <parent link="base_link"/>
      <child link="link1"/>
      <axis xyz="0 0 1"/>
      <limit lower="0" upper="5.3" effort="0" velocity="0"/>
</joint>
```

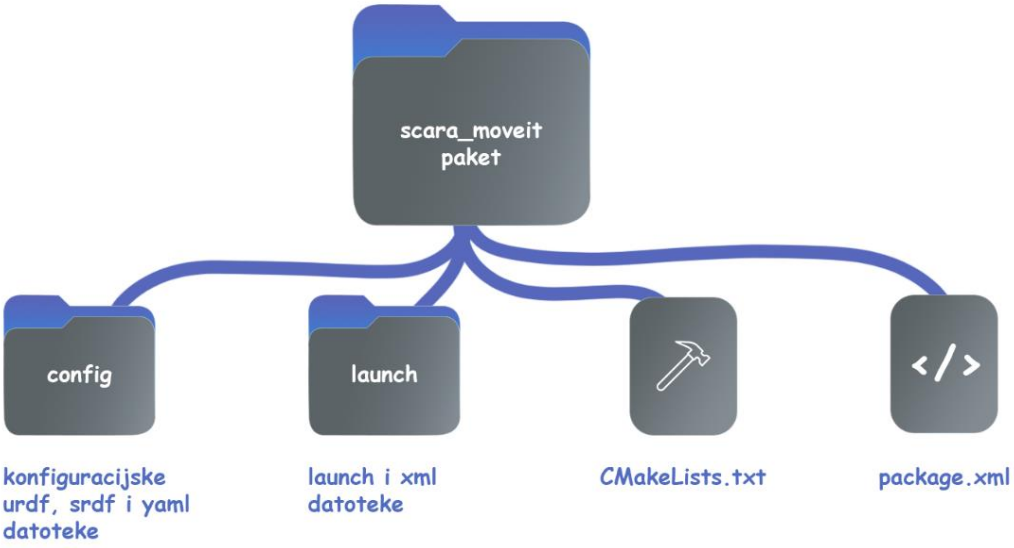
Kako bi se iz postojećeg 3D modela generirala URDF datoteka, koristi se ekstenzija za programski paket *SolidWorks*. Prije generiranja datoteke preporučeno je pojednostavljenje geometrije. Unutar ekstenzije odabiru se zglobovi i odgovarajući segmenti. Prvo se odabire baza robota, odnosno fiksni nepokretni segment kojem se definira ime te odabire broj segmenata koji se veže na njega. Nakon odabira sljedećeg segmenta definira se vrsta i ime zgloba između odabranog segmenta i prethodnog. Kako je prvi zglob rotacijski, odabire se vrsta zgloba *revolute* kojoj se za razliku od *continuos* zgloba, definiraju krajnje vrijednosti. Sljedeći zglob je translacijski zglob, odnosno *prismatic*, kojem se kao i rotacijskom zglobu definiraju krajnje vrijednosti. Treći i četvrti zglob su također rotacijski te se ponavlja isti postupak kao i kod prvoga zgloba. Nakon definiranja svih zglobova ruke, definira se hvataljka koja se sastoji od fiksnog dijela te dva prsta. Između hvataljke i ruke definira se fiksni zglob, odnosno *fixed*. Kontrola oba prsta preko jednoga zgloba postiže se opcijom *Mimic other joint.* Nakon definiranja svih imena segmenata, zglobova i krajnjih vrijednosti, ekstenzija zatim kreira koordinatne sustave i osi rotacije svih zglobova. Prije završnih koraka potrebno je provjeriti jesu li generirani koordinatni sustavi točno generirani te ako nisu ispraviti greške. Zatim se generira URDF datoteka.



Slika 5.4. Prozor konfiguratora zglobova

5.2. ROS paketi

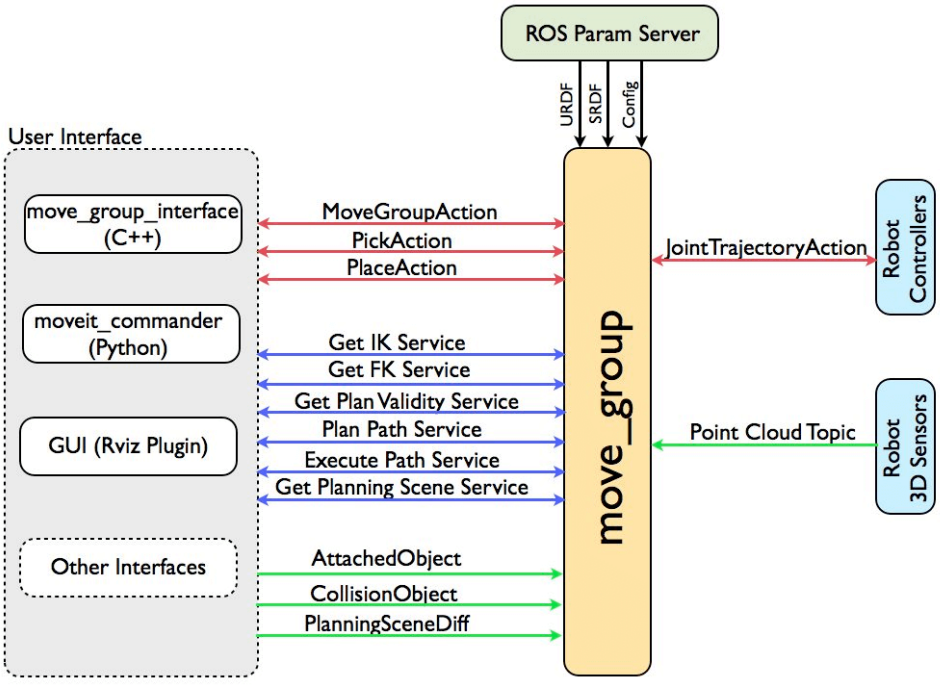
Nakon instalacije ROS-a, kreira se radni prostor unutar kojega se nalazi skup direktorija i datoteka koje se koriste za organizaciju i upravljanje paketima. Prilikom stvaranja radnog prostora unutar terminala, automatski se stvaraju direktoriji *build, devel* i *src*. *Build* direktorij služi za konfiguriranje i izgradnju paketa, *devel* služi za kompilaciju svih izvršnih datoteka i biblioteka koje se pozivaju komandom *catkin_make*, dok je *src* direktorij unutar kojega se pohranjuju svi korisnički paketi. ROS paketi su način organiziranja čvorova, biblioteka, podataka ili bilo čega što može biti koristan modul.



Slika 5.5. Hijerarhija *scara_moveit* paketa

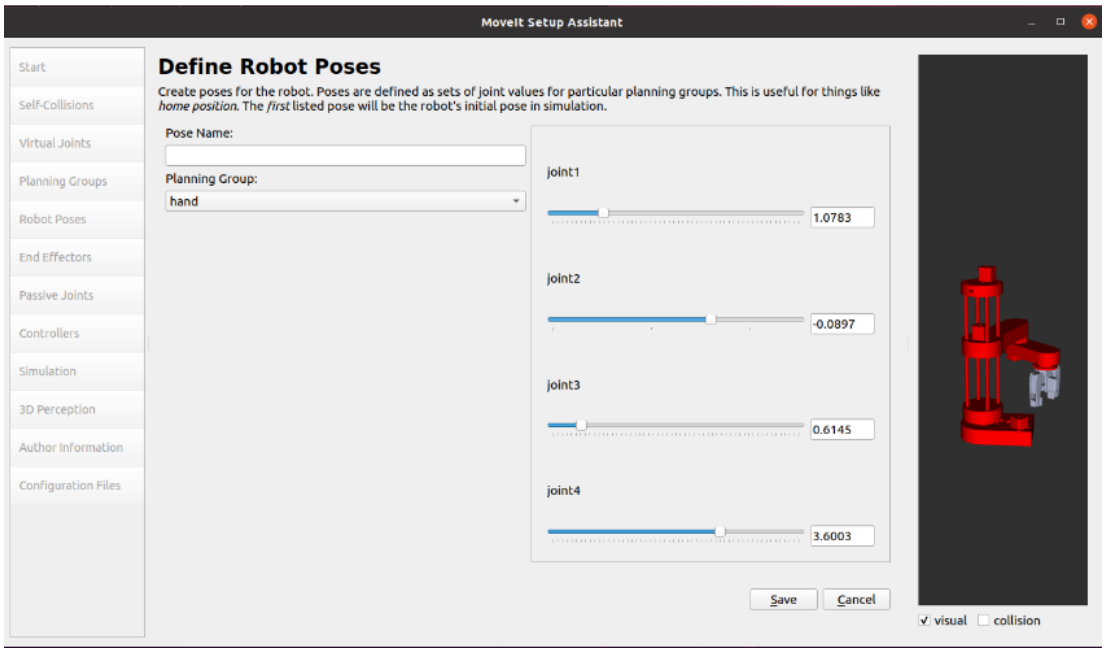
5.3. MoveIt!

MoveIt! je ROS-ov softverski paket korišten za manipulaciju, planiranje kretanja, kinematiku, 3D percepciju, kontrolu i navigaciju. Omogućuje planiranje putanja kreiranjem matrica kolizija na temelju URDF datoteke. Primarni čvor kojeg MoveIt pruža je *move*_group čija je funkcija osiguravanje komunikacije pojedinačnih komponenti ROS-a kako bi se izvršile željene akcije. Kako bi se tim akcijama *move_group* se može pristupiti pomoću C++ *move_group_interface* paketa, Pyhton *moveit_commander* paketa ili pomoću ROS-ovog grafičkog sučelja *Rviz*. *Move_group* dobiva informacije od URDF i SRDF datoteka te od MoveIt konfiguracije pomoću koje se definira kinematika, planiranje gibanja, percepcija, kontrola i navigacija. Pomoću Robot Interface-a, *move_group* komunicira s robotom kako bi odredio trenutno stanje robota, kao što su pozicije zglobova, statusi senzora i oblaci točaka, preko ROS tema *joint_state* i *tf*, preko ROS akcijskih sučelja te raznih ekstenzija. Pomoću ekstenzijskog sučelja *MoveIt!* komunicira i koristi razne planere kretnji koje osiguravaju izvedive kretnje bez kolizija.



Slika 5.6. *move_group* čvor [14]

U sklopu ovoga rada, *MoveIt!* se koristi za upravljanje fizičkim i simuliranim robotom. Pomoću *MoveIt Setup Assistant*-a kreira se paket s potrebnim konfiguracijskim datotekama, parametrima i *launch* datotekama koje pomažu integrirati *MoveIt!* sa fizičkim robotom. Pri pokretanju *MoveIt Setup Assistant*-a odabire se opcija kreiranja paketa na temelju URDF datoteke kreirane *SolidWorks* ekstenzijom. Nakon toga kreira se matrica kolizija pomoću koje se testiraju zglobovi koji bi mogli međusobno doći u koliziju. Ti položaji se zabranjuju te se tako smanjuje vrijeme potrebno za planiranje kretnji. Zatim se definiraju virtualni zglobovi te glavne grupe robota. Glavne grupe robota dijele se na ruku robota i hvataljku. Prilikom definiranja grupe ruke robota definira se i kinematički rješavač, pomoću kojega *MoveIt!* rješava problem inverzne kinematike, kako bi kretnje robota bile usklađene. Definiraju se i predefinirane poze robota pomoću kojih se kasnije robot postavlja u željeni položaj. Sljedeći korak je odabir prihvatnice. Odabire se grupa kojoj prihvatnica pripada te zglob na koji je pričvršćena. Pasivni zglobovi su svi zglobovi na čije kretanje utječe neki drugi zglob. U slučaju SCARA robota to su prsti hvataljke pokretani pomoću servomotora. Pridodaju se kontroleri koji služe za pomicanje grupa robota pomoću ROS *Control* paketa te se na kraju imenuje i generira ROS paket s novim konfiguracijskim URDF, SRDF i *launch* datotekama.



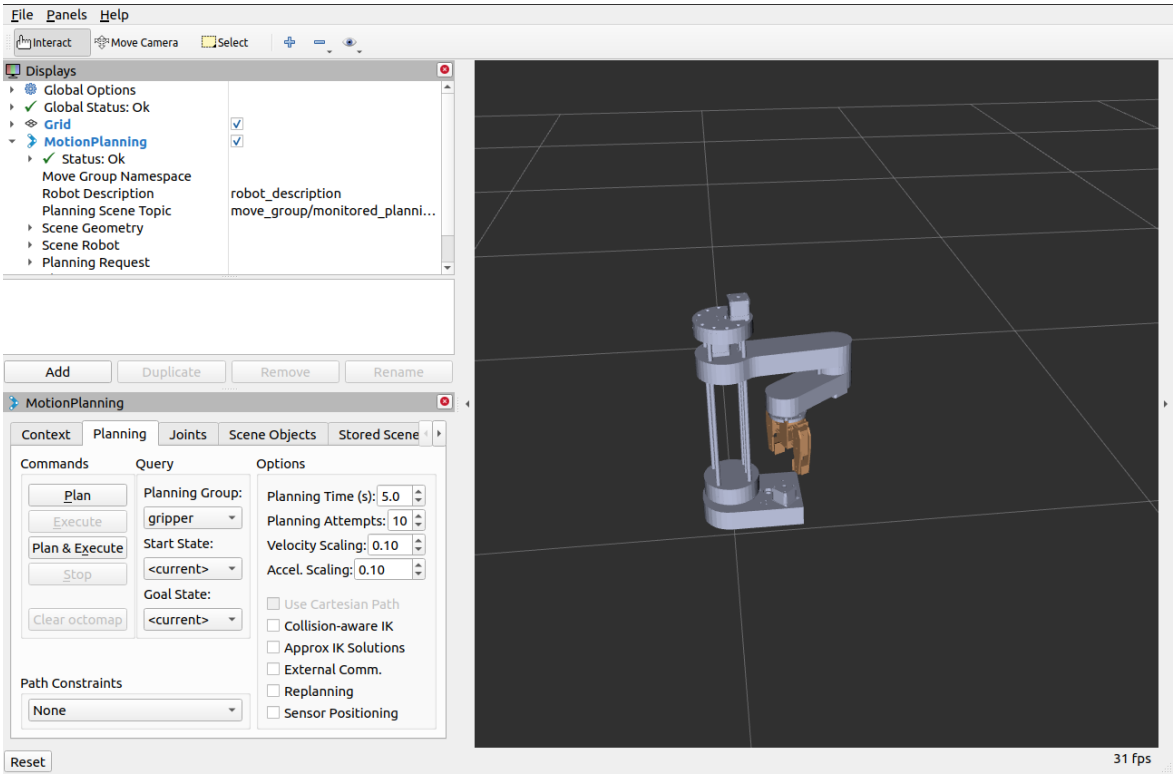
Slika 5.7. MoveIt Setup Assistant

launch datoteke su datoteke XML formata koje definiraju konfiguraciju i ponašanje ROS čvorova i ostalih komponenata ROS sustava. Omogućuju pokretanje više čvorova i *launch* datoteka istovremeno, namještanje parametara, pokretanje i namještanje ROS tema te tako olakšavaju konfiguraciju složenijih robotskih sustava. U sklopu ovoga rada *launch* datoteka se nadopunjuje XML elementom <*node*>, argumenata ime, paket, vrsta i izlaz, pomoću kojeg se, uz pokretanje simulacijskog okruženja i ROS tema, pokreće čvor za pretvorbu informacija. Dana je skraćena verzija *demo.launch* datoteke korištene za pokretanje RViz-a i čvorova u sklopu ovoga rada.

```
<launch>
  <arg name="pipeline" default="ompl" />
  <!--Ostali arg elementi...-->
  <arg name="use_rviz" default="true" />
  <node pkg="tf2_ros" type="static_transform_publisher"
name="virtual_joint_broadcaster_0" args="0 0 0 0 0 0 world base_link" />
  <group if="$(eval arg('moveit_controller_manager') == 'fake')">
    <node name="joint_state_publisher" pkg="joint_state_publisher"
type="joint_state_publisher" unless="$(arg use_gui)">
      <rosparam
param="source_list">[move_group/fake_controller_joint_states]</rosparam>
    </node>
    <node name="joint_state_publisher" pkg="joint_state_publisher_gui"
type="joint_state_publisher_gui" if="$(arg use_gui)">
      <rosparam
param="source_list">[move_group/fake_controller_joint_states]</rosparam>
    </node>
    <node name="robot_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher" respawn="true" output="screen" />
  </group>
  <node name="scara_control_node" pkg="scara_control" type="control_node.py"
output="screen"/>
  <include file="$(dirname)/move_group.launch">
    <arg name="allow_trajectory_execution" value="true"/>
    <!--Ostali arg elementi...-->
    <arg name="load_robot_description" value="$(arg load_robot_description)"/>
  </include>
  <include file="$(dirname)/moveit_rviz.launch" if="$(arg use_rviz)">
    <arg name="rviz_config" value="$(dirname)/moveit.rviz"/>
    <arg name="debug" value="$(arg debug)"/>
  </include>
  <include file="$(dirname)/default_warehouse_db.launch" if="$(arg db)">
    <arg name="moveit_warehouse_database_path" value="$(arg db_path)"/>
  </include>
</launch>
```

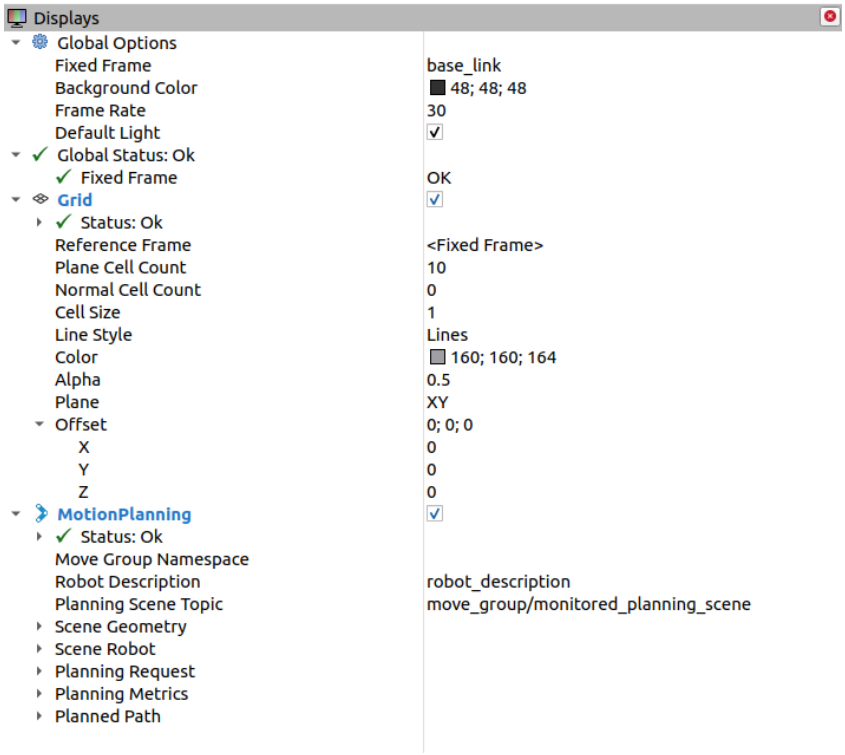
5.4. RViz

RViz je ROS-ovo grafičko sučelje za 3D vizualizaciju robotskih sustava. Omogućuje grafički prikaz vanjskih informacija pomoću simuliranog 3D modela pravoga robota te na isti način i slanje informacija stvarnome robotu pomoću ROS tema. Glavna tema koja se koristi za komunikaciju između simulacijskog robota i pravog je *joint_states*. *joint_states* je ROS tema koja sadrži informacije o imenima zglobova, njihovim pozicijama, brzini i momentu. U sklopu ovoga rada, pomoću *RViz*-a i *Motion Plannera* kontrolira se stvarni robot.



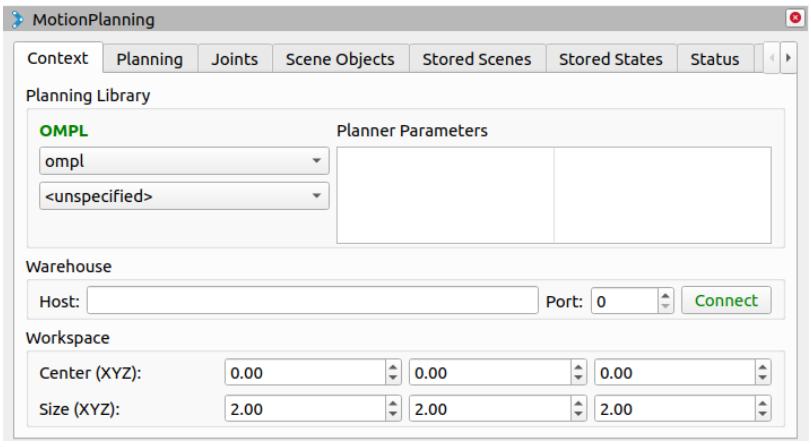
Slika 5.8. SCARA robot unutar *RViz*-a

Unutar *Displays* prozora *RViz*-a nalaze se sve opcije koje utječu na 3D svijet, kao što su robot, oblaci točaka i koordinatne osi. Globalne opcije služe za glavne definicije simuliranog svijeta. *Fixed frame* je koordinatni sustav u koji se svi ostali koordinatni sustavi transformiraju. *Global Status* javlja postoje li greške unutar definiranih parametara *displays* liste. Kao referentni *frame* postavlja se *Fixed frame* koji je u ovome radu bazni segment robota. *MotionPlanning* prikazuje sve informacije vezane uz scenu planiranja kretnji te omogućava animirani prikaz simuliranih kretnji robota. *Robot Description* služi za lociranje i učitavanje URDF datoteke robota te je njegova zadana vrijednost *robot_description* koja nastaje prilikom kreiranja URDF datoteke pomoću *SolidWorks* ekstenzije ili *MoveIt Setup Assistant*-a. Unutar *MotionPlanning*-a mogu se definirati vizuali simulacije.



Slika 5.9. *Displays* prozor *RViz*-a

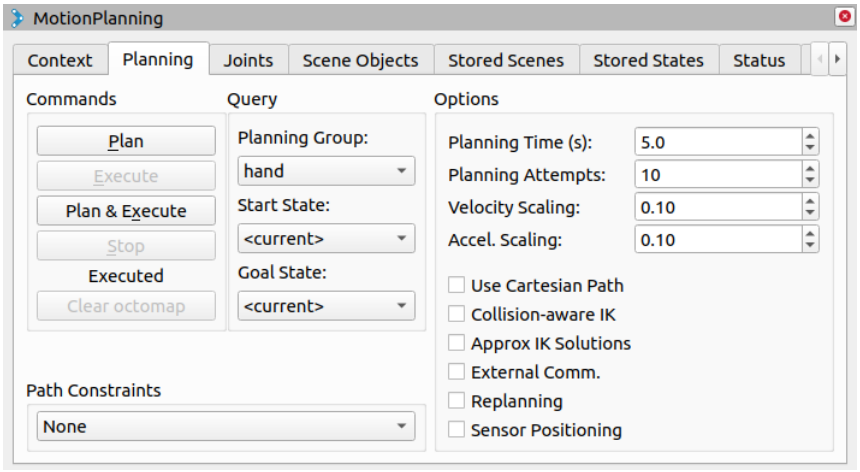
MotionPlanning također ima svoj prozor u kojemu se mogu definirati parametri simulacije. Kao *Planning Library* koristi se *ompl* biblioteka otvorenog koda.



Slika 5.10. *MotionPlanning* prozor *RViz*-a

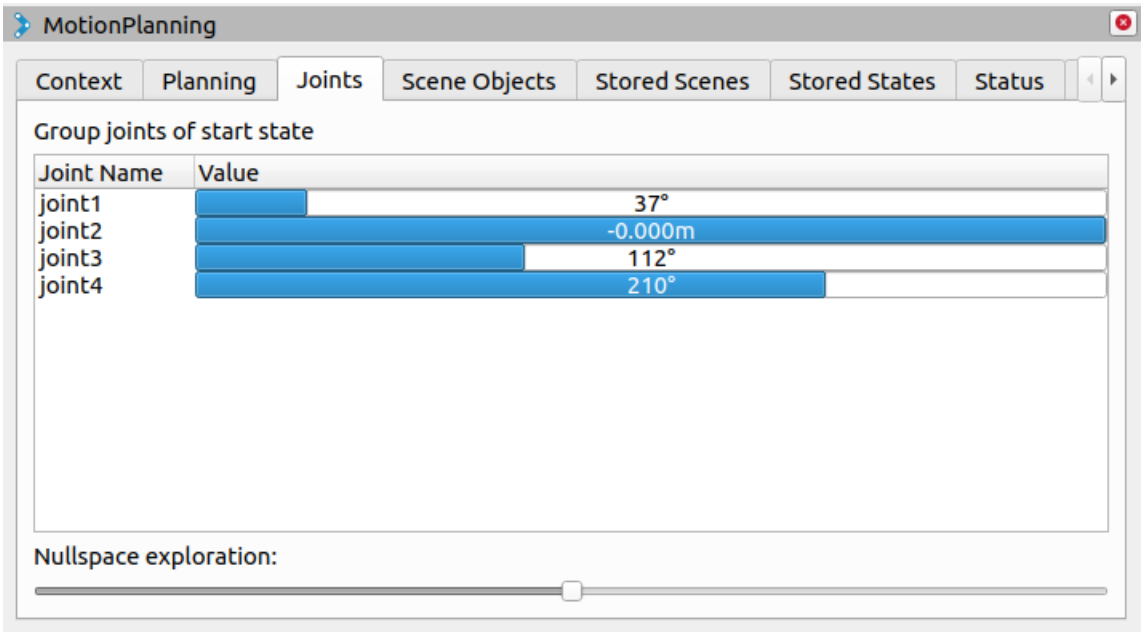
Unutar *Planning* kartice planiraju se te izvode kretnje. Odabire se grupa definirana u *MoveItSetupAssistant*-u koja će provoditi kretnje, početni položaj (*Start State*) i krajnji

položaj (*Goal State*) te počinje planiranje. *RViz* planiranu kretnju pokazuje sjenovitim modelom robota. Ukoliko se žele postići linearne kretnje robota odabire se opcija *Use Cartesian Path*. Za izbjegavanje prepreka koristi se opcija *Collision-aware IK*. Kao i ROS, *RViz* koristi teme za kontrolu robota te se može omogućiti vanjska kontrola kretnji robota opcijom *External Comm.*



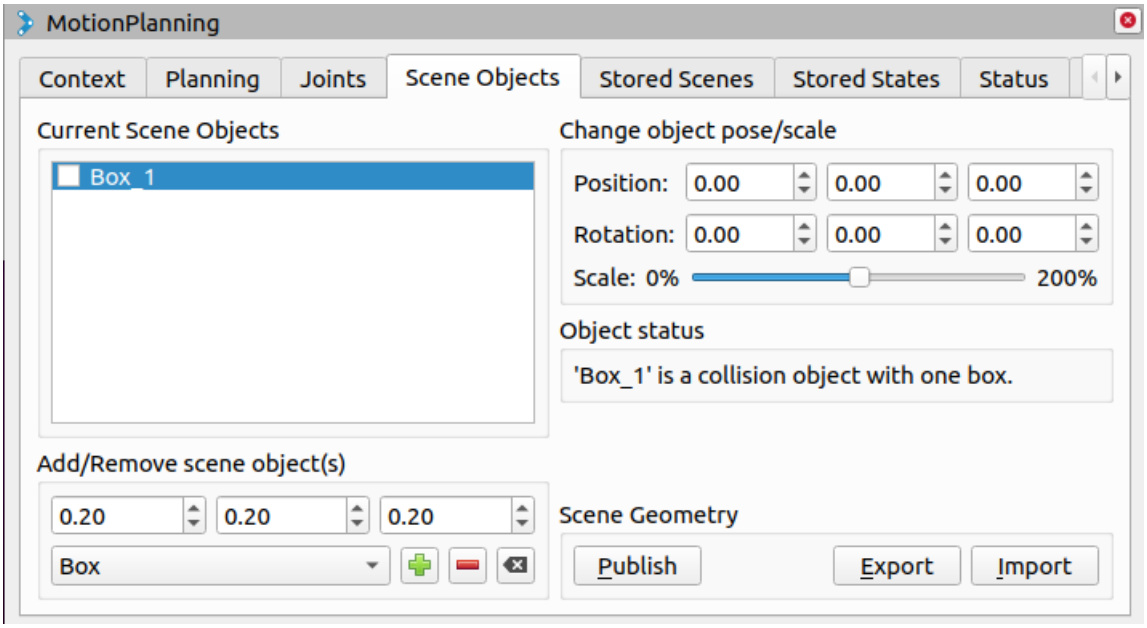
Slika 5.11. *MotionPlanning Planning* kartica *RViz*-a

Direktna kontrola svakoga zgloba zasebno postiže se u *Joints* kartici *MotionPlanner*-a. Prikazani su svi zglobovi odabrane grupe te klizači njihovih vrijednosti. U kombinaciji sa postavljanjem *Goal State*-a na vrijednost *current* mogu se definirati željena krajnja pozicija robota.



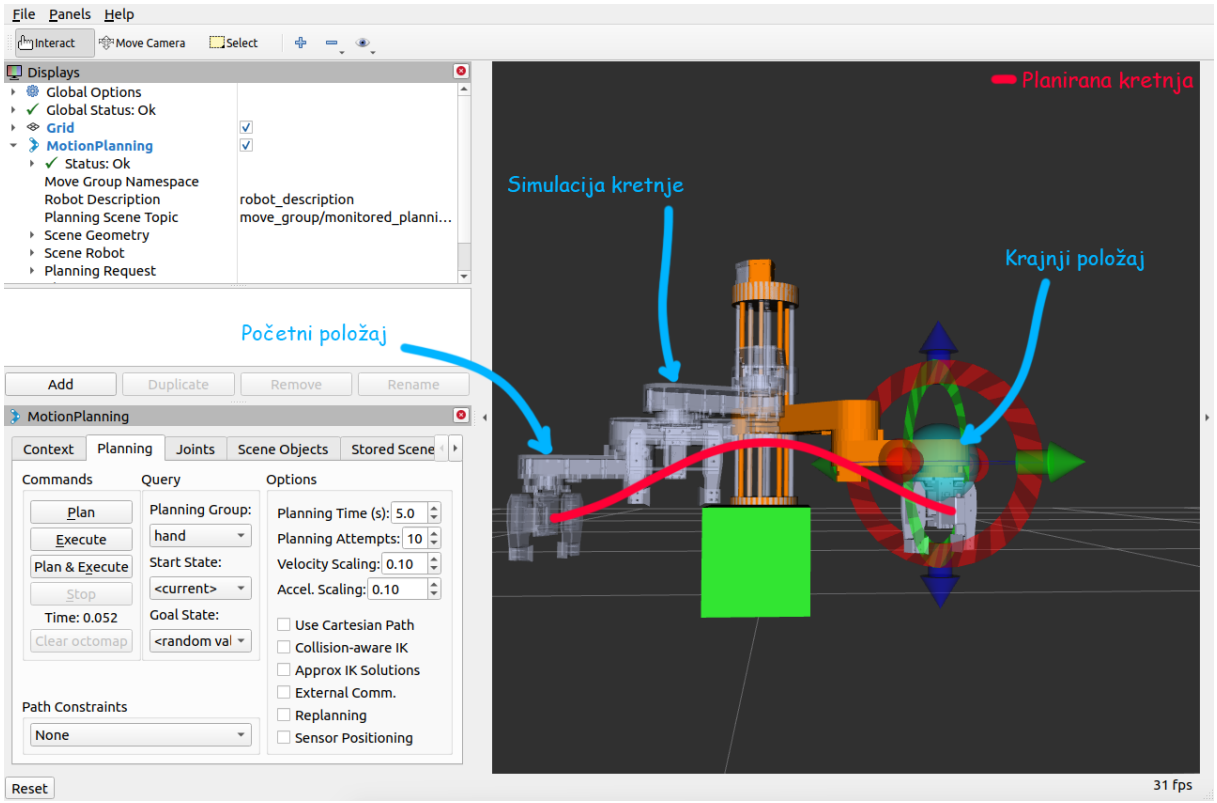
Slika 5.12. *MotionPlanning Joints* kartica *RViz*-a

Mogućnost dodavanja objekata, kao što su prepreke ili okolina, nalazi se u kartici *Scene Objects*. Nakon što je objekt dodan, može se izmijeniti njegova veličina i rotacija te provjeriti je li u kolizija s nekim drugim objektom.



Slika 5.13. *MotionPlanning Scene Objects* kartica *RViz*-a

RViz zatim dodani objekt uzima u obzir pri planiranju kretnji.



Slika 5.14. Izbjegavanje kolizije s dodanim objektom

5.5. control_node.py

Pomoću *control_node.py* čvora, kreira se ROS tema koja konvertira prihvaćene informacije o položaju s teme *joint_states* u prikladne informacije za kretnje robota. Nove kretnje robota zapisuju se u novu *msg* datoteku te se prosljeđuju na temu *joint_steps* na koju je pretplaćen Arduino. Iako se podaci mogu procesirati direktno u Arduinu, *control_node.py* napravljen je sa svrhom lakšeg podešavanja transformacija kutova *joint_states* teme u korake koračnih motora. Dan je isječak koda *control_node.py* unutar kojega se inicijalizira čvor, pomoću komande *rospy.init_node()*, pretplatnik na temu *joint_states* komandom *rospy.Subscriber()* koja za parametre koristi ime teme, *msg* datoteku i funkciju koju poziva prilikom dolaska novih podataka te pošiljatelj na novu temu *joint_steps* komandom *rospy.Publisher()*, čiji su parametri također ime teme, *msg* datoteka i veličina reda poslanih podataka. Kako bi se kod neprestano izvodio i konstantno slao i primao nove podatke koristi se *while* petlja s uvjetom *not rospy.is_shutdown()*, odnosno statusom rada ROS servera.

```
def main():
    global joint_status

    rospy.init_node("scara_control")
    rospy.Subscriber("/joint_states", JointState, cmd_cb)
    pub = rospy.Publisher("joint_steps", FinalJoints, queue_size=50)

    rate = rospy.Rate(20)

    while not rospy.is_shutdown():
        if joint_status == 1:
            joint_status = 0
            pub.publish(total)
            rate = rospy.Rate(20)
        rate.sleep()
```

5.6. ROS i Arduino

Kako bi se ostvarila komunikacija između ROS-a i Arduina, potrebno je instalirati *rosserial_arduino* ROS-ov paket. *rosserial_arduino* omogućuje mikro kontroleru pretplatu i objavljivanje na željene ROS teme te instalaciju potrebnih biblioteka za iste. Kako bi se omogućila kontrola koračnih motora, instalira se i *AccelStepper* biblioteka unutar Arduinovog softvera za programiranje Arduino IDE. Pošto koračni motori nemaju enkodere, pomoću *AccelStepper* prate se trenutne vrijednosti njihovih pozicija te se na temelju njih obavljaju daljnje instrukcije dobivene od *joint_steps*.

Unutar Arduino-vog koda potrebno je definirati sve skripte, pinove i varijable koje se koriste. Primanje i slanje podataka između Arduina i ROS-a inicijalizira se pomoću *ros::NodeHandle nh;* komande koja omogućava stvaranje pretplatnika i pošiljatelja unutar Arduina te održava komunikaciju serijskog porta. Isječak *arduino_sketch.ino* koda za definiranje skripti, pinova i varijabli te inicijalizacija komunikacije između ROS-a i Arduina:

```
#include <ros.h>
#include <scara_control/FinalJoints.h>
#include <Servo.h>
#include <std_msgs/Bool.h>
#include <std_msgs/String.h>
#include <math.h>
#include <std_msgs/Int16.h>
#include <std_msgs/UInt16.h>
#include <AccelStepper.h>
#include <MultiStepper.h>

AccelStepper joint1(1, 2, 5);
AccelStepper joint2(1, 12, 13);
AccelStepper joint3(1, 4, 7);
AccelStepper joint4(1, 3, 6);

Servo gripper;

const int limitSwitchPins[] = {11, A3, 9, 10};
int joint_step[5];
int joint_positions[5];
int joint_status = 0;
int homingStepValue = 1;
ros::NodeHandle nh;
std_msgs::Int16 msg;
```

Prije definiranja pretplatnika potrebno je definirati funkcije koje će pretplatnici pozivati prilikom pristizanja novih podataka. Zatim se definira pretplatnik na temu *joint_steps* komandom (*ros::Subscriber<scara_control::FinalJoints> arm_sub("joint_steps",arm_cb);*). Prilikom pokretanja Arduina poziva se *setup()* funkcija. Unutar *setup()* funkcije potrebno je inicijalizirati čvor, komandom *nh.initNode()*, te prethodno definirane pretplatnike, komandom *nh.subscribe(ime_teme)*. Zatim se postavljaju vrijednosti brzina i akceleracija koračnih motora te poziva *homing()* funkcija. Svrha *homing()* funkcije je postavljanje svih zglobova u nulti položaj, odnosno položaj u kojemu su svi mikro prekidači aktivirani. Dani su isječci koda *arduino_sketch.ino* funkcije *setup()* vezani uz inicijalizaciju čvora te definiranje svih motora i pozivne funkcije *arm_cb()* koja zapisuje podatke dobivene od *joint_steps* teme.

Isječak *setup()* funkcije:

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

joint1.setMaxSpeed(1500);
joint2.setMaxSpeed(2000);
joint3.setMaxSpeed(2000);
joint4.setMaxSpeed(2000);

joint1.setAcceleration(2000);
joint2.setAcceleration(2000);
joint3.setAcceleration(2000);
joint4.setAcceleration(2000);

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

gripper.attach(A0);
```

arm_cb() pozivna funkcija pretplatnika *arm_sub*:

```
void arm_cb(const scara_control::FinalJoints& arm_steps){
  joint_status = 1;
  joint_step[0] = arm_steps.position1;
  joint_step[1] = - arm_steps.position2;
  joint_step[2] = arm_steps.position3;
  joint_step[3] = - arm_steps.position4;
  joint_step[4] = arm_steps.position5;
}
```

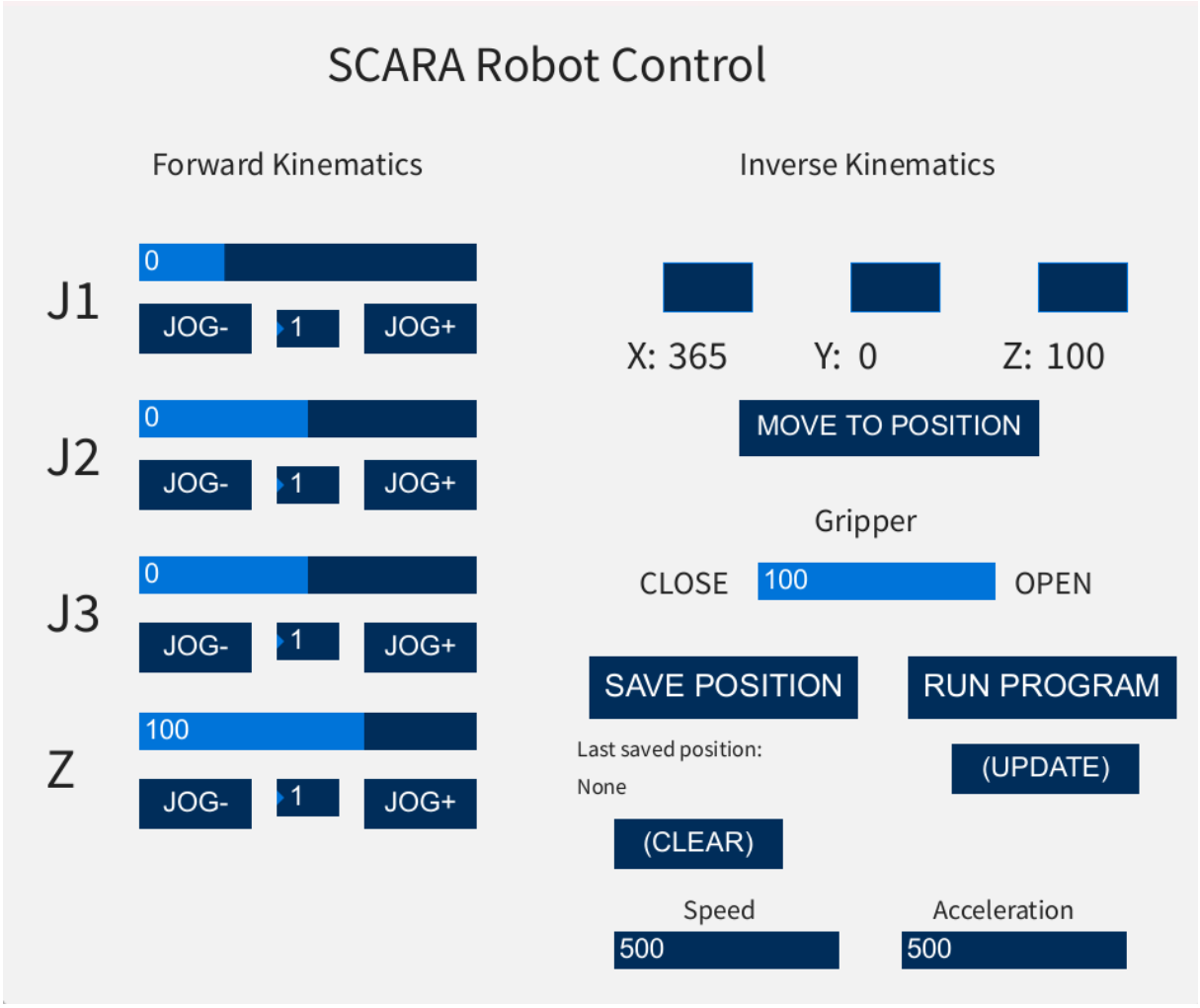
Nakon izvršavanja *setup()* funkcije Arduino poziva *loop()* funkciju koja se izvršava u petlji. Unutar *loop()* funkcije procesiraju se podaci poslani na pretplatničke funkcije te provode zadane kretnje. Da bi podaci bili konstanto procesirani poziva se funkcija *nh.spinOnce()* zadužena za obradu podataka ROS čvorova.

loop() funkcija:

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

    joint1.setSpeed(400);
    joint2.setSpeed(400);
    joint3.setSpeed(400);
    joint4.setSpeed(400);
    joint1.setAcceleration(200);
    joint2.setAcceleration(200);
    joint3.setAcceleration(200);
    joint4.setAcceleration(200);

    joint1.moveTo(positions[0]);
    joint2.moveTo(positions[1]);
    joint3.moveTo(positions[2]);
    joint4.moveTo(positions[3]);

    while (joint1.currentPosition() != positions[0] ||
joint2.currentPosition() != positions[1] || joint3.currentPosition() !=
positions[2] || joint4.currentPosition() != positions[3]) {
      joint1.run();
      joint2.run();
      joint3.run();
      joint4.run();
  }
  delay(100);
    gripper.write(joint_step[5]);
  }
  joint_status = 0;

  nh.spinOnce();
  delay(1);
}
```

6. INTEGRACIJA I TESTIRANJE

Uz 3D model robota, How To Mechatronics prilaže i Arduinov kod te grafičko sučelje za upravljanje robotom. Upravljanje robotom postiže se direktnom ili inverznom kinematikom. U grafičko sučelje upisuju se vrijednosti kutova zglobova ili *x y* i *z* koordinate, odabire se status hvataljke te se pokreće program. Sučelje također nudi opciju namještanja brzine i akceleracije koračnih motora, spremanje pozicija te pokretanje ponavljajućeg programa na temelju spremljenih pozicija.



Slika 6.1. Grafičko sučelje [1]

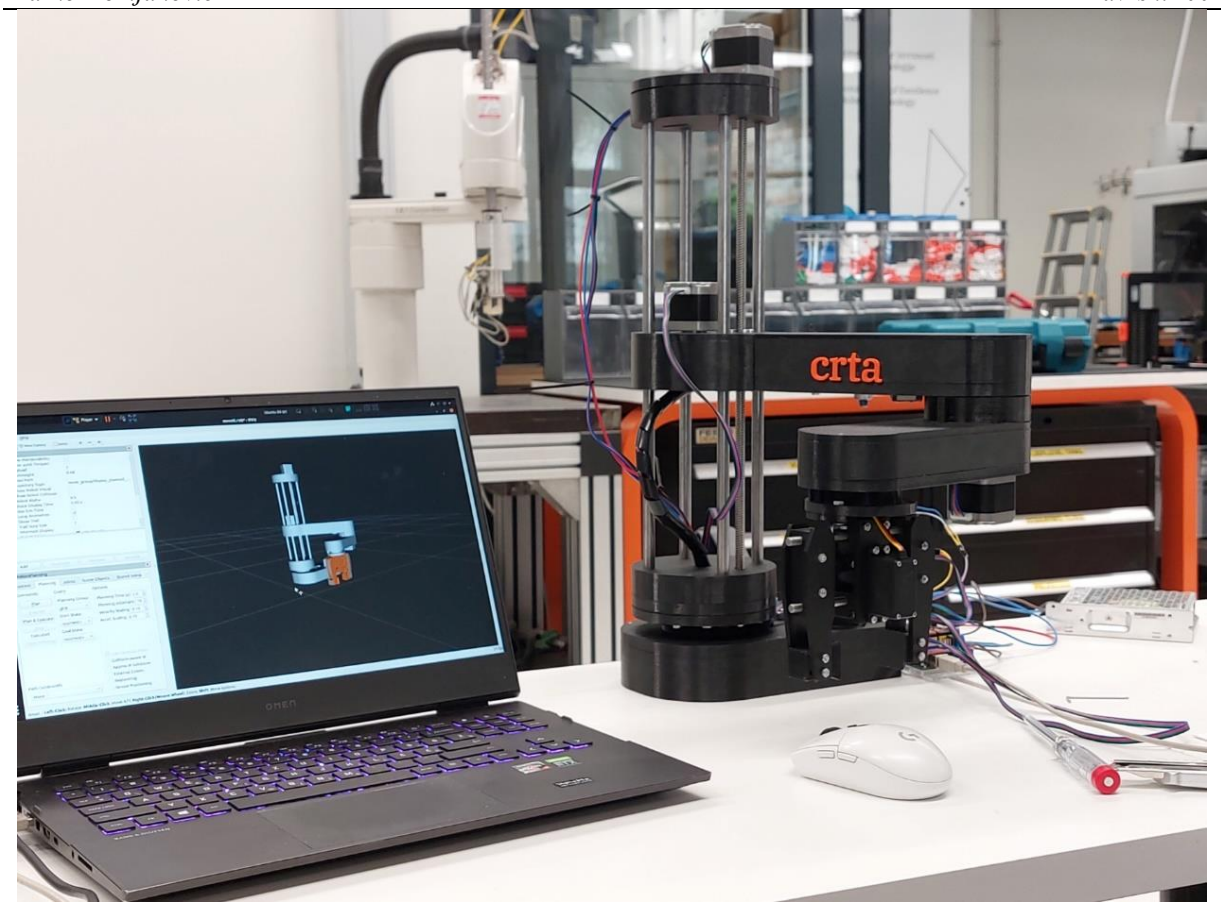
ROS pruža grafičko sučelje *RViz* pomoću kojega se kontroliraju zglobovi simulacijskog i stvarnog robota. Također se može upravljati direktnom i inverznom kinematikom. Direktna kinematika se izvodi pomoću terminala. Inverzna kinematika se računa prilikom postavljanja interaktivnog koordinatnog sustava vrha alata. Pomoću simulacijskog modela stvara se uvid u stanje stvarnoga robota. ROS planira najbolje kretnje između točaka te izbjegava kolizije. Brzine zglobova se namještaju unutar koda Arduina. Fino namještanje zglobova se postiže se sa *control_node.py*. Svi ROS paketi, programski kodovi i URDF datoteke nalaze se na repozitoriju GitHub-a [15]. Pokreće se *demo.launch* datoteka koja postavlja sve potrebne teme te otvara grafičko sučelje *RViz* komandom *roslaunch scara_moveit demo.launch*. Nakon uspješnog pokretanja *RViz*-a, provjerava se uspješnost pokretanja *control_node.py* istom *launch* datotekom tako da se provjeri popis tema te postoji li *joint_steps* tema. Komandom *rostopic list* prikazuje se popis tema u terminalu. Kako bi se ostvarila komunikacija između Arduina i ROS-a potrebno je pokrenuti serijsku komunikaciju pomoću *rosserial*-a. *rosserial* se pokreće komandom *rosrun rosserial_arduino serial_node.py _port:=/dev/ttyAMC0 _baud:=57600* kojoj je potrebno specificirati serijski port na kojemu je Arduino povezan sa računalom.

```
Loading 'pilz_industrial_motion_planner/MoveGroupSequenceService'...
[ INFO] [1694955543.750438471]: Reading limits from namespace /robot_description_planning
[ INFO] [1694955543.762651194]:

*****************************************************
* MoveGroup using:
*     - ApplyPlanningSceneService
*     - ClearOctomapService
*     - CartesianPathService
*     - ExecuteTrajectoryAction
*     - GetPlanningSceneService
*     - KinematicsService
*     - MoveAction
*     - PickPlaceAction
*     - MotionPlanService
*     - QueryPlannersService
*     - StateValidationService
*     - SequenceAction
*     - SequenceService
*****************************************************

[ INFO] [1694955543.763536837]: MoveGroup context using planning plugin ompl_interface/OMPLPlanner
[ INFO] [1694955543.763586370]: MoveGroup context initialization complete

You can start planning now!
```

Slika 6.2. Pokretanje *RViz*-a

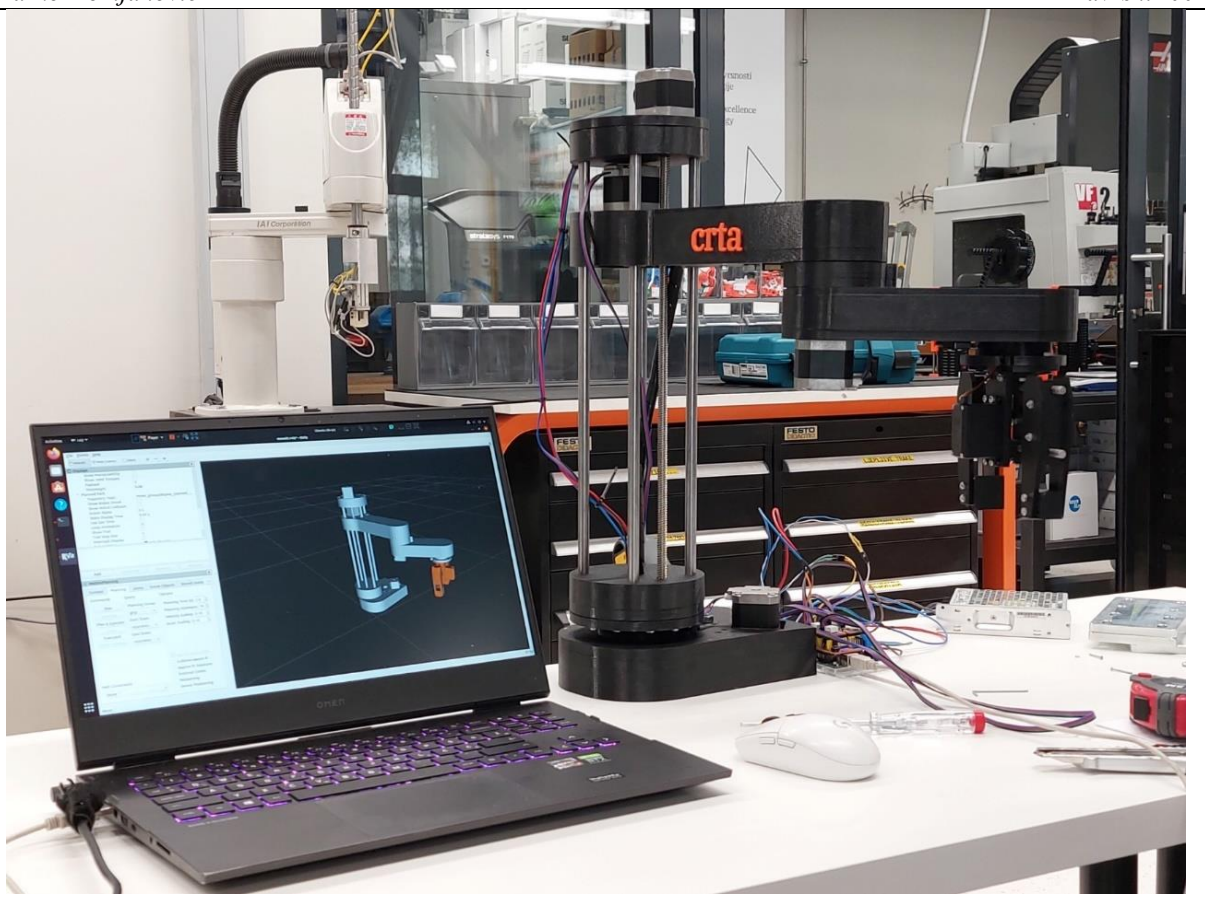
```
mingo@ubuntu:~/catkin_ws$ rosrun rosserial_arduino serial_node.py _port:=/dev/ttyACM0 _baud:=57600
[INFO] [1694956019.461794]: ROS Serial Python Node
[INFO] [1694956019.464427]: Connecting to /dev/ttyACM0 at 57600 baud
[INFO] [1694956021.569910]: Requesting topics...
[INFO] [1694956025.690944]: Note: subscribe buffer size is 512 bytes
[INFO] [1694956025.692130]: Setup subscriber on joint_steps [scara_control/FinalJoints]
```

Slika 6.3. Ostvarena komunikacija Arduina i ROS-a



Slika 6.4. *RViz* i SCARA – položaj 1

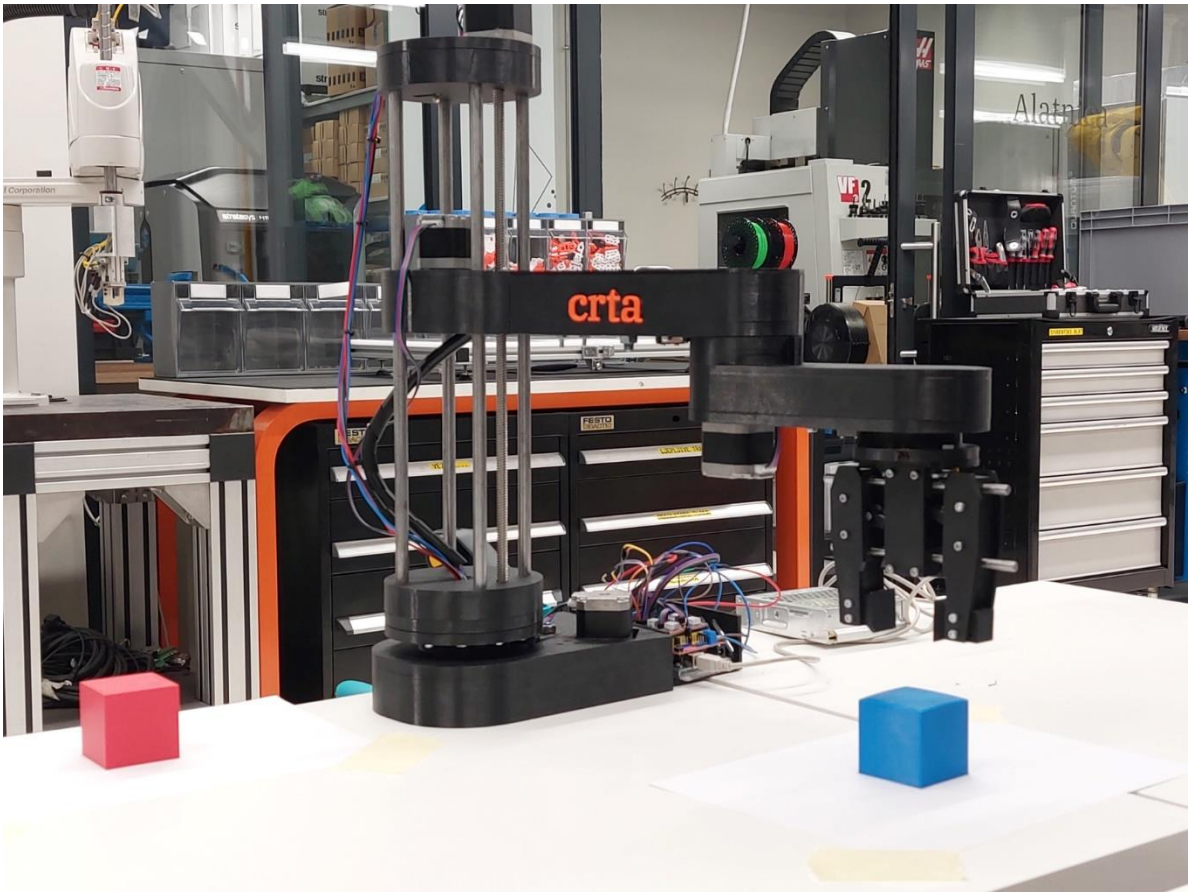
Kružne kretnje postižu se namještanjem klizača vrijednosti kutova zglobova te pritiskanjem na gumb „*Plan&Execute*“, dok se linearna gibanja postižu aktiviranjem opcije „*Use Cartesian path“* te namještanjem vrha alata pomoću interaktivnog markera.



Slika 6.5. *RViz* i SCARA – položaj 2

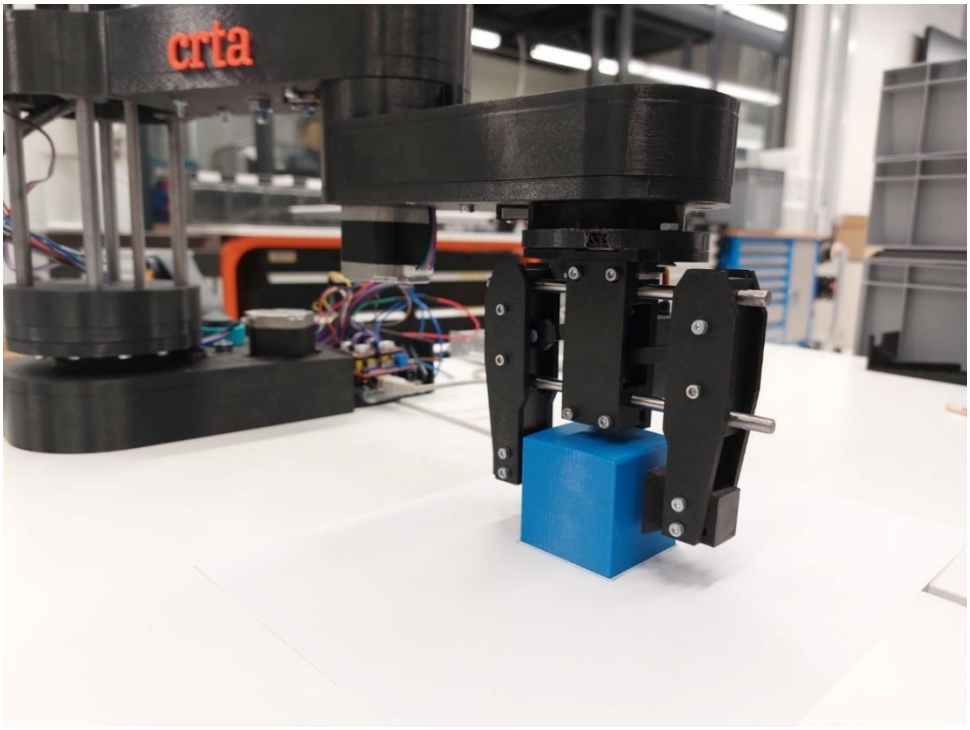
6.1. Validacija

Kako bi se validirala funkcionalnost sustava, robotu se zadaje zadatak. Pokreću se potrebni čvorovi te se pomoću *RViz*-a upravlja robotom. Najčešći zadatak SCARA robota je „pick and place“, pa je zato odabran upravo taj zadatak. Postavlja se eksperimentalno okruženje u kojem se provjerava preciznost i ponovljivost.



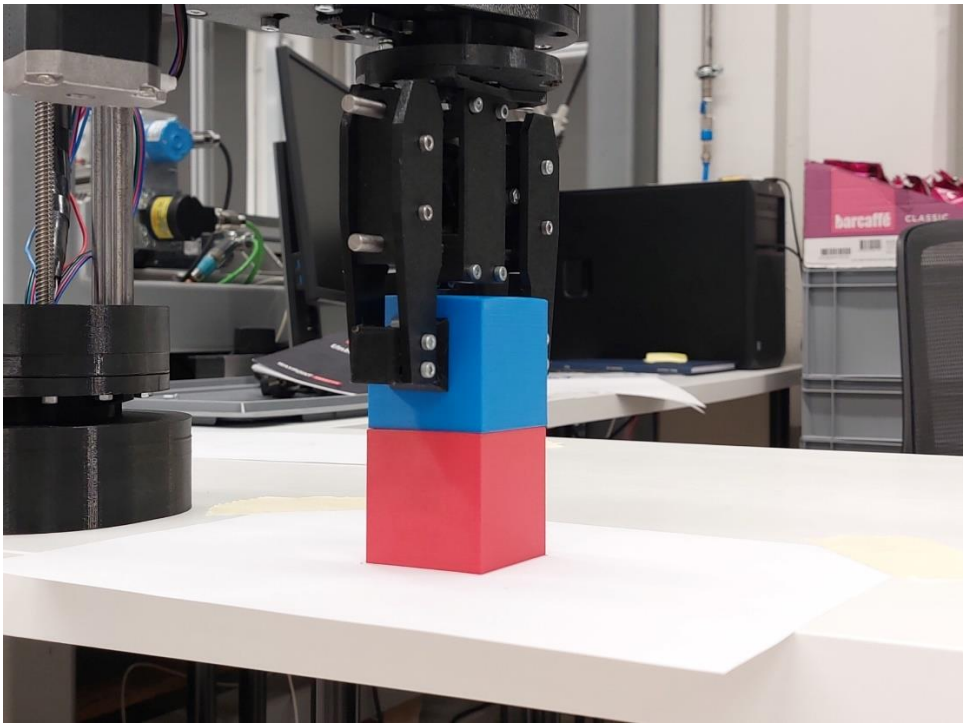
Slika 6.6. Eksperimentalno okruženje

Zadatak robota je podići kocku i postaviti ju na drugu kocku te ponoviti proces nekoliko puta. Robot se prvo postavlja u položaj pripreme hvata iznad kocke pomoću kružnih kretnji te se zatim linearno spušta i prihvaća kocku.



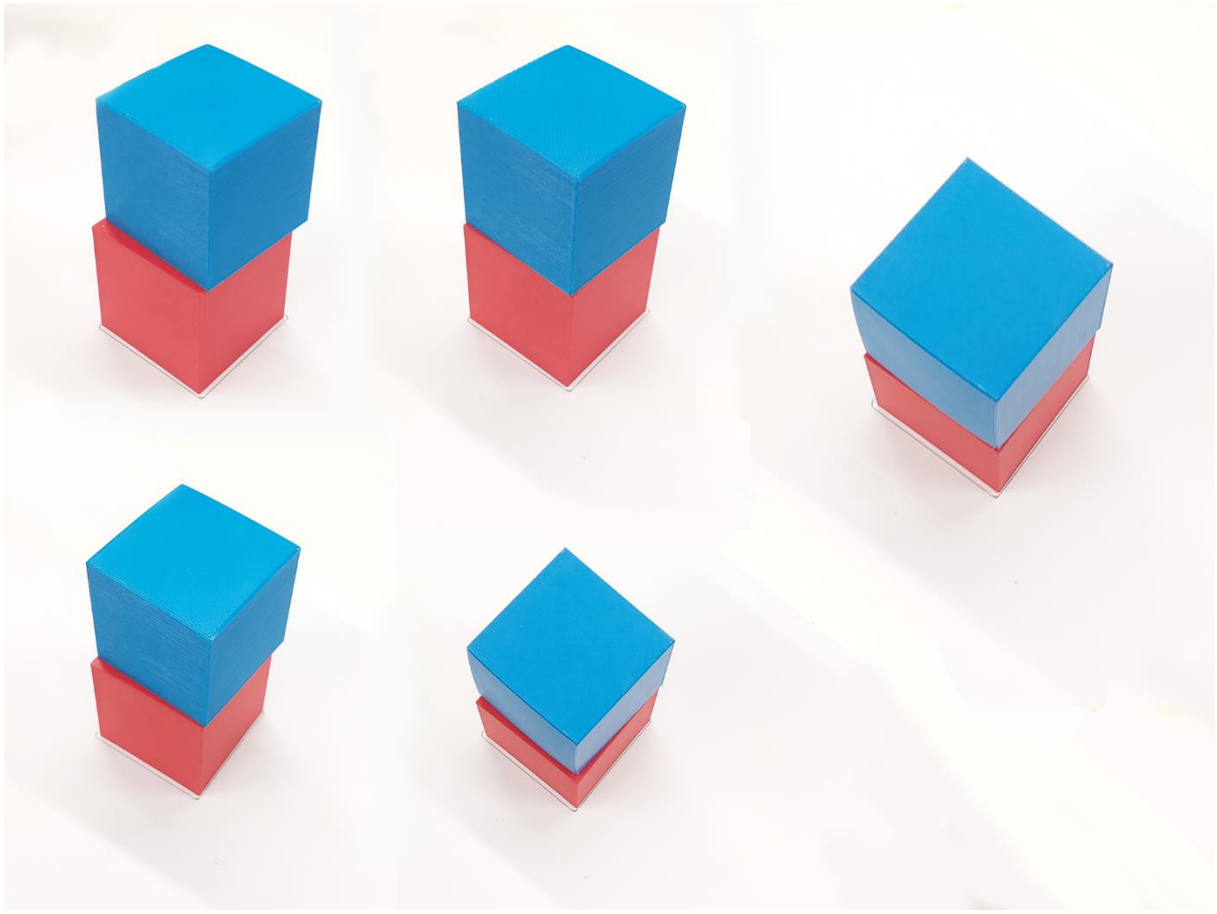
Slika 6.7. Prihvaćanje kocke

Nakon što robot prihvati kocku linearno se translatira po z-osi te kružnim gibanjem dolazi iznad druge kocke. Kako bi se provjerila točnost i ponovljivost, pomoću ROS-a robotu su svaki put zadane iste vrijednosti rotacija i translacija zglobova.



Slika 6.8. Odlaganje kocke

Rezultati eksperimentalne validacije, temeljeni na pet ponavljanja, vidljivi su na slici 6.9. Na temelju prijenosnih omjera, postavki mikro koraka koračnih motora te duljine segmenata, izračunate su greške rotacijskih zglobova koje iznose 0,982 milimetara u baznom zglobu te 0,117 milimetara u zglobu robotske ruke. Kako bi se postigla veća preciznost, potrebno je koristiti veće prijenosne omjere ili drivere s mogućnošću manjih postavki mikro koraka, kao što je STSPIN200.



Slika 6.9. Rezultati „pick and place“

7. ZAKLJUČAK

SCARA roboti su odlična podloga za učenje robotike koja je sve zastupljenija u industriji i svijetu. Kako bi inženjeri stekli potrebna znanja i vještine, javlja se potreba za edukacijskim varijantama industrijskih robota. Edukacijski robot mora biti što pristupačniji, a neki od načina povećanja pristupačnosti su 3D tiskanje dijelova te razvoj vlastitog upravljačkog sustava. ROS je besplatno, svima dostupno i vrlo moćno okruženje opremljeno raznim alatima korištenim u svrhu upravljanja, kako već postojećih tako i novih robota. Iako ima strmiju krivulju učenja, zbog svoje široke dostupnosti postoje zajednice i velik broj postojećih rješenja pomoću kojih se lakše nalazi rješenje vlastitom problemu.

LITERATURA

[1] https://howtomechatronics.com/projects/scara-robot-how-to-build-your-own-arduino-based-robot/ [10.02.23.]

[2] https://new.abb.com/products/robotics/robots/scara-robots/irb-910sc [03.08.23.]

[3] https://www.fanucamerica.com/products/robots/series/scara/sr-6ia-scara-robot [03.08.23.]

[4] https://epson.com/For-Work/Robots/SCARA/Epson-LS3-SCARA-Robots---400mm/p/RLS3401ST9P5 [03.08.23.]

[5] http://www.robothalloffame.org/inductees/06inductees/scara.html [05.08.23.]

[6] https://prusament.com/materials/prusament-petg/ [06.08.23.]

[7] Ivan Virgala, Michal Kelemen, Alexander Gmiterko, and Tomáš Lipták, "Control of Stepper Motor by Microcontroller." Journal of Automation and Control, vol. 3, no. 3 (2015): 131-134. doi: 10.12691/automation-3-3-19.

[8] https://store.arduino.cc/products/arduino-mega-2560-rev3 [09.08.23.]

[9] Y. M. Hasan, L. F. Shakir and H. H. Naji, "Implementation and Manufacturing of a 3-Axes Plotter Machine by Arduino and CNC Shield," *2018 International Conference on Engineering Technology and their Applications (IICETA)*, Al-Najaf, Iraq, 2018, pp. 25-29, doi: 10.1109/IICETA.2018.8458071.

[10] Serdar Kucuk and Zafer Bingul (2006). Robot Kinematics: Forward and Inverse Kinematics, Industrial Robotics: Theory, Modelling and Control, Sam Cubero (Ed.), ISBN: 3-86611-285-8, InTech, Available from: http://www.intechopen.com/books/industrial_robotics_theory_modelling_and_control/robot_kinematics__forwar d_and_inverse_kinematics

[11] http://wiki.ros.org/ROS/ [15.08.23.]

[12] http://wiki.ros.org/rosserial_arduino [18.08.23.]

[13] https://gachiemchiep.github.io/ros_srs/013_urdf_1/ [20.08.23.]

[14] https://moveit.ros.org/documentation/concepts/ [15.08.23.]

[15] https://github.com/Domingo9732/ROS-Educational-SCARA

PRILOZI

I. Arduino-v kod

II. control_node.py

III. https://github.com/Domingo9732/ROS-Educational-SCARA

PRILOG 1: ARDUINO-V KOD

```
#include <ros.h>
#include <scara_control/FinalJoints.h>
#include <Servo.h>
#include <std_msgs/Bool.h>
#include <std_msgs/String.h>
#include <math.h>
#include <std_msgs/Int16.h>
#include <std_msgs/UInt16.h>
#include <AccelStepper.h>
#include <MultiStepper.h>


AccelStepper joint1(1, 2, 5);
AccelStepper joint2(1, 12, 13);
AccelStepper joint3(1, 4, 7);
AccelStepper joint4(1, 3, 6);

Servo gripper;
MultiStepper steppers;

const int limitSwitchPins[] = {11, A3, 9, 10};

int joint_step[5];
int joint_positions[5];
int joint_status = 0;
int homingStepValue = 1;
ros::NodeHandle nh;
std_msgs::Int16 msg;

void homing() {
  homingStepValue = -1;
  while(digitalRead(11) != 1){
    joint1.moveTo(homingStepValue);
    homingStepValue--;
    joint1.run();
  }

  homingStepValue = -1;
  while(digitalRead(A3) != 1){
    joint2.moveTo(homingStepValue);
    homingStepValue--;
    joint2.run();
    delay(1);
  }


  homingStepValue = -1;
```

```
  while(digitalRead(10) != 1){
    joint3.moveTo(homingStepValue);
    homingStepValue--;
    joint3.run();
    delay(1);
  }

  homingStepValue = 1;
  while(digitalRead(9) != 1){
    joint4.moveTo(homingStepValue);
    homingStepValue++;
    joint4.run();
    delay(1);
  }


  joint1.setCurrentPosition(0);
  joint2.setCurrentPosition(0);
  joint3.setCurrentPosition(0);
  joint4.setCurrentPosition(0);
}

void arm_cb(const scara_control::FinalJoints& arm_steps){
  joint_status = 1;
  joint_step[0] = arm_steps.position1;
  joint_step[1] = - arm_steps.position2;
  joint_step[2] = arm_steps.position3;
  joint_step[3] = - arm_steps.position4;
  joint_step[4] = arm_steps.position5;

}

ros::Subscriber<scara_control::FinalJoints> arm_sub("joint_steps",arm_cb);

void setup() {
  //put your setup code here, to run once:
  nh.getHardware()->setBaud(57600);
  Serial.begin(115200);
  joint_status = 0;
  pinMode(11, INPUT_PULLUP);
  pinMode(10, INPUT_PULLUP);
  pinMode(9, INPUT_PULLUP);
  pinMode(A3, INPUT_PULLUP);

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

  joint1.setMaxSpeed(1500);
  joint2.setMaxSpeed(2000);
```

```
  joint3.setMaxSpeed(2000);
  joint4.setMaxSpeed(2000);

  joint1.setAcceleration(2000);
  joint2.setAcceleration(2000);
  joint3.setAcceleration(2000);
  joint4.setAcceleration(2000);

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

  gripper.attach(A0);
  delay(20);
  homing();
  delay(50);

}

void loop() {
  if (joint_status == 1)
  {
    long positions[5];

    positions[0] = joint_step[0];
    positions[1] = joint_step[1];
    positions[2] = -joint_step[2];
    positions[3] = -joint_step[3];
    positions[4] = joint_step[4];

    joint1.setSpeed(400);
    joint2.setSpeed(400);
    joint3.setSpeed(400);
    joint4.setSpeed(400);

    joint1.setAcceleration(200);
    joint2.setAcceleration(200);
    joint3.setAcceleration(200);
    joint4.setAcceleration(200);

    joint1.moveTo(positions[0]);
    joint2.moveTo(positions[1]);
    joint3.moveTo(positions[3]);
    joint4.moveTo(positions[2]);

    while (joint1.currentPosition() != positions[0] ||
joint2.currentPosition() != positions[1] || joint3.currentPosition() !=
positions[3] || joint4.currentPosition() != positions[2]) {
```

```
      joint1.run();
      joint2.run();
      joint3.run();
      joint4.run();
    gripper.write(positions[4]);
    }
  }

  joint_status = 0;
  nh.spinOnce();
  delay(1);

}
```

PRILOG 2: CONTROL_NODE.PY

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

import rospy
from sensor_msgs.msg import JointState
from scara_control.msg import FinalJoints
import math

arm_steps = FinalJoints()
total = FinalJoints()
steps = [12000, 25000, 14200, 4500]
joint_status = 0
cur_angle = [0, 0, 0, 0]
joint_step = [0, 0, 0, 0]
prev_angle = [0, 0, 0, 0]
init_angle = [0, 0, 0, 0]
total_steps = [0, 0, 0, 0]
count = 0

def cmd_cb(cmd_arm):
    global count
    global prev_angle
    global init_angle

    if count == 0:
        prev_angle = cmd_arm.position[:4]
        init_angle = cmd_arm.position[:4]

    for i in range(4):
        joint_step[i] = int((cmd_arm.position[i] - prev_angle[i]) * steps[i] /
(2 * math.pi))

    arm_steps.position1 = joint_step[0]
    arm_steps.position2 = joint_step[1] * 25
    arm_steps.position3 = joint_step[2]
    arm_steps.position4 = joint_step[3]

    if cmd_arm.position[4] > 0.012:
        arm_steps.position5 = 180
    else:
        arm_steps.position5 = 0

    if count != 0:
        prev_angle = cmd_arm.position[:4]

    total.position1 += arm_steps.position1
    total.position2 += arm_steps.position2
    total.position3 += arm_steps.position3
```

```
    total.position4 += arm_steps.position4
    total.position5 = arm_steps.position5

    global joint_status
    joint_status = 1
    count = 1

def main():
    global joint_status

    rospy.init_node("scara_control")
    rospy.Subscriber("/joint_states", JointState, cmd_cb)
    pub = rospy.Publisher("joint_steps", FinalJoints, queue_size=50)

    rate = rospy.Rate(50)

    while not rospy.is_shutdown():
        if joint_status == 1:
            joint_status = 0
            pub.publish(total)
            rate = rospy.Rate(50)  # Reset the rate for the next cycle
        rate.sleep()  # Sleep to control the loop rate


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